USER MANUAL

Version FlexSim 7.1.2

Date Published
17 MAR 2014

Table of Contents

Welcome To FlexSim
Whats New
Getting Started

- Getting Started with FlexSim and Simulation
- Interacting With FlexSim
- Keyboard Interaction
- FlexSim Terminology
- First Model
- License Activation
    - License Activation Concepts
    - License Activation Example
    - License Activation Reference

FlexSim Concepts

- FlexSim Concepts Overview
- Flowitems
- Ports
- Order of Events
- Itemtype
- Labels
- Item and Current
- Return Values
- Picklists
- Template Code
- Model Tree View

Tutorials

- Tutorials Introduction
- Lesson 1 Tutorial
    - Introduction
    - Step-By-Step Model Construction
- Lesson 2 Tutorial
    - Introduction
    - Step-By-Step Model Construction
- Lesson 2 Extra Mile
    - Introduction
    - Step-By-Step Model Construction
- Lesson 3 Tutorial
    - Introduction
    - Step-By-Step Model Construction
- Labels Tutorial
    - Introduction
    - Step-By-Step Model Construction
- Global Modeling Tools Tutorial
    - Introduction
    - Step-By-Step Model Construction
- User Events Tutorial
    - Introduction
    - Step-By-Step Model Construction
- TimeTables Tutorial
    - Introduction
    - Step-By-Step Model Construction
- Kinematics Tutorial
    - Introduction
    - Step-By-Step Model Construction
- Task Sequence Tutorial 1
    - Introduction
    - Step-By-Step Model Construction
- Task Sequence Tutorial 2
    - Introduction
    - Step-By-Step Model Construction
- Task Sequence Tutorial 3
    - Introduction
    - Step-By-Step Model Construction
- SQL Tutorial
    - Introduction
    - Step-By-Step Model Construction
- Fluid Objects Tutorial
    - Introduction
    - Step-By-Step Model Construction

Modeling Views

- Orthographic/Perspective View
- Tree Window
- Travel Networks Utility
- Library Icon Grid
- View Settings
- Light Source Editor
- Quick Properties
- Edit Selected Objects
- Find Objects
- Groups
- Model Layouts
- Measure / Convert

Main Menu and Toolbar

- File Menu
- Edit Menu
- View Menu
- Build Menu
- Execute Menu
- Statistics Menu
- Tools Menu
- Debug Menu
- Help Menu
- FlexSim Toolbar
- Simulation Run Panel

General Windows

- Attribute Hints
- Model Settings
- Global Preferences Window
- Tree Browse Dialog
- Database Table View
- Table Editor
- Find and Replace

Object Windows

- Object Properties Windows Overview
- FixedResource Properties Pages
  - BasicFR Advanced Page
  - Breakdowns Page
  - Combiner Page
  - Conveyor Page
  - Decision Points Page
  - Flow Page
  - Layout Page
  - MergeSort Flow Page
  - MultiProcessor Page
  - Photo Eyes Page
  - Processor Page
  - ProcessTimes Page
  - Queue Page
  - Rack Page
  - Separator Page
  - Sink Page
  - SizeTable Page
  - Source Page
- TaskExecuter Properties Pages
  - ASRSvehicle Page
  - BasicTE Page
  - Breaks Page
  - Collision Page
  - Crane Page
  - Dispatcher Page
  - Geometry Page
  - Robot Page
  - TaskExecuter Page
  - Transporter Pages
- Fluid Properties Pages
  - Blender Page
  - FluidConveyor Page
  - FluidLevelDisplay Page
  - FluidProcessor Page
  - FluidToItem Page
  - Generator Page
  - Initial Product Window
  - Inputs/Outputs Page
  - ItemToFluid Page
  - Marks Page
  - Mixer Page
  - Percents Page
  - Pipe Page
  - Pipe Layout Page
  - Recipe Page
  - Sensors Page
  - Splitter Page
  - Steps Page
  - Tank Page
  - Terminator Page
  - Ticker Page
- Shared Properties Pages
  - General Page
  - Labels Page
  - Triggers Page
  - Statistics Window
- Other Properties Pages
  - Container Page
  - Container Functionality Page
  - Display Page
  - NetworkNode Page
  - NetworkNodes Page
  - Speeds Page
  - Traffic Control Page

FlexSim Object Library

- FlexSim Object Library Overview
- FixedResources
  - FixedResources Concepts
- TaskExecuters
  - TaskExecuters Concepts
- Travel Networks
  - NetworkNode
  - TrafficControl
- VisualTool
  - VisualTool Overview
  - VisualTool Example
- Fluid Library
  - Fluid Library Concepts

Modeling Tools

- Animation Creator
  - Animation Creator Concepts
  - Animation Creator Example
  - Animation Creator Reference
- AVI Maker
  - AVI Maker Concepts
  - AVI Maker Example
- Event List
- Event Log
- Excel Interface
- Flowitem Bin

    - Flowitem Bin Concepts
    - Flowitem Bin Reference
- Global Tables
- Global Task Sequences
    - Global Task Sequences Concepts
    - Global Task Sequences Example
- Global Variables
- Graphical User Interfaces
    - Graphical User Interfaces Concepts
    - Graphical User Interfaces Example
    - Graphical User Interfaces Reference
- Media Files
- Model Background
- Model Triggers
- MTBF/MTTR
- Presentation Builder
- Script Console
- Time Tables
    - Time Tables Concepts
    - Time Tables Reference
- Tracked Variables
- User Commands
    - User Commands Concepts
    - User Commands Reference
- User Events
- Microsoft Visioâ„¢ Importer

Pick Lists

- Triggers
    - Breakdown/Repair Trigger (On Break Down/On Repair)
    - Collision Trigger (Handle Collision)
    - Creation Trigger (On Creation)
    - Down/Up Trigger (On Break Down/On Repair)
    - Entry/Exit Trigger (On Entry/On Exit)
    - Load/Unload Trigger (On Load/On Unload)
    - Message Trigger (On Message)
    - Node Entry Trigger (On Continue/On Arrival)
    - OnChange Trigger
    - OnCover OnUncover Trigger
    - OnDraw Trigger (Custom Draw Code)
    - OnEmpty / OnFull Trigger
    - OnEntryRequest Trigger
    - OnReceiveTaskSequence
    - OnResourceAvailable Trigger
    - Setup/Process Finish Trigger (On Setup Finish/On Process Finish)
    - Reset Trigger (On Reset)
- Time Pick Lists
    - Load/Unload Time
    - Minimum Staytime (Minimum Dwell Time)
    - Process Time
    - Setup Time
    - Time Picklist (Inter-Arrivaltime Usage)
- FixedResources Picklists
    - Flow Rate
    - Item Speed (Speed)
    - On Clear
    - On Cover
    - Pick Operator
    - Place in Bay
    - Place in Level
    - Pull Requirement
    - Pull Strategy
    - Rise/Fall Through Mark Triggers
    - Send Requirement
    - Send To Port
    - Split Quantity (Split/Unpack Quantity)
    - Transport Dispatcher (Request Transport From)
- TaskExecuters Picklists
    - Break To (Break To Requirement)
    - Load / Unload Time
    - Pass To
    - Queue Strategy
- Experimentation Picklists
    - End of Experiment
    - End of Run (End of Replication)
    - End of Scenario
    - End of Warmup (End of Warmup Period)
    - Performance Measure
    - Start of Experiment
    - Start of Run (Start of Replication)
    - Start of Scenario
- Other Picklists
    - Text Display

Task Sequences

- Task Sequences Concepts
- Custom Built Task Sequences
- Task Sequence Preempting
- Coordinated Task Sequences
- Task Sequences Quick Reference
- Task Sequence Types
- Querying Information on Task Sequences

Charting and Reporting

- Dashboard
  - Dashboard Concepts
  - Custom Chart
  - Model Input
  - Dashboard Example
  - Dashboard Reference
  - Dashboard Graphs
    - Associations Page
    - Colors Page
    - Data Page
    - Date and Time Display
    - Financial Objects Page
    - General Pages
    - HTML Statistic (Model Documentation)
    - Item Trace Page
    - Objects Page
    - Statistics Page
    - Tracked Variables
    - Utilization Analysis Page
  - Model Input Properties
- Reports and Statistics

FlexSim Coding

- Writing Logic in FlexSim
- Basic Modeling Functions and Logic Statements
- Code Editor
- Debugging
  - Debugging Overview
  - Breakpoints
  - Call Stack
  - Code Profiler
  - Local Variables
  - Watch Variables
- Command Helper

Experimenter Optimizer

- Experimenter
- Optimization in FlexSim
- Experimenter/ Optimizer Example
- Experimenter / Optimizer Reference

3D Media

- Importing 3D Media
- Preparing a 3D File
- Importing AutoCAD Drawings
- Shape Factors
- Shape Frames
- Level Of Detail (LOD)

Miscellaneous Concepts

- Advanced Undo
  - Advanced Undo Concepts
  - Advanced Undo Example
- Custom Libraries
  - Custom Libraries Concepts
  - ModelLibraries Node
  - Custom Libraries Example
- FlexSim Tree Structure
- FlexSim XML
- GUI Events and View Attributes
- View Attributes Reference
- Kinematics
  - Kinematics Concepts
  - Kinematics Commands
- Sampler
  - Sampler Concepts
  - Referencing a Label Example
  - Referencing a Global Table Example
  - Assigning a Transporter Example
  - Changing Item Routing Example
- SQL Queries
  - SQL Queries Concepts
  - SQL Queries Example
  - SQL Queries Reference
- State List
- Webserver
  - Webserver Concepts
  - Webserver Example
- When To Compile FlexSim

Welcome to the FlexSim User Manual

If this is your first time using FlexSim, try some of the following suggestions:

1. Go to FlexSim's YouTube Channel and watch how to build your very first model.

2. Read the Getting Started section of the User Manual and follow along by building the model described.

3. Work through the Tutorials described here in the User Manual (only the first two can be completed with the Trial Version).

Have some fun by building your own model from scratch!

Feel free to contact our technical support staff if you have any questions while evaluating the software. FlexSim Technical Support can be reached Monday - Friday, 8:00 am - 5:00 pm MST. You may call 801-224-6914, e-mail your questions using a web form at http://www.flexsim.com/support/, or post your questions on our worldwide FlexSim Community Forum (http://www.flexsim.com/community/forum/). We hope you enjoy learning how FlexSim can help you optimize your flow processes.

What's New in FlexSim

The following is a list of features and fixes that have been included in the latest FlexSim releases.

FlexSim 7.1.2

- Fixed a memory leak with the 3D view.
- Fixed a bug in the MTBF/MTTR that caused the downtime value in the down function to always be 0.
- Fixed a bug causing Keyframe triggers to not be fired in operator animations.
- Fixed Animation Editor > Draw Surrogate > Main Object Content.
- Fixed Animation Editor > Component > Rotational Centroid fields.
- Fixed issues with editing animations in models where units were not meters.
- Fixed an issue causing global variables to be renamed to NULL when creating multiple global variables at the same time.
- Fixed exceptions being thrown when you try to use shaders in compatibility mode.
- Fixed issues displaying skp files while using the A* Module.
- Fixed an issue with not being able to select views in the GUI builder.
- Fixed DLL loading to properly load DLLs from the model directory.

FlexSim 7.1.0

- Added Financial Analysis Dashboard Widget.
- Added Custom Dashboard Widget to allow any numeric data to be displayed in the dashboard, including table data, bundle data, global variables, etc.
- Added State Gantt Chart Dashboard Widget.
- Added a FlowItem Trace Gantt Chart Dashboard Widget.
- Dashboard table data can display current state values as strings.
- Added functionality to export dashboards to HTML.
- Moved the statistics tab out of object properties windows and into the Quick Properties window.
- Labels, tables, statistics, global variables and tracked variables can be "pinned" to the dashboard.
- More options to customize Dashboard Widgets (font size, bar size, custom display names, etc).
- More Model Input objects in the Dashboard including Radio Buttons, Listboxes, Trackers and Tables.
- Added picklist options for starting and stopping animations.
- Added picklist option to display labels on FlowItems in the 3D view.
- Moved the User Manual into FlexSim as a dockable window.
- Redesigned Robot GUI.
- Animation variables can point to components in an animation for quick referencing, not just surrogates.
- Improved User Toolbar items for Dashboard, GlobalTables, TimeTables and MTBF/MTTR buttons to allow for opening the objects.
- Improvements to shadows, shaders and mesh drawing.
- Improvements to the SKP Reader.
- Removed Model Views utility and placed it in the Quick Properties window.
- Added a "Headlight" feature to light sources.
- Updated Excel Importer/Exporter to handle relative paths of workbooks to the model.
- Improved the Excel Import's excelreadstr() and Automatic Data Distinction to more accurately read in values from excel, including dates and times.
- Added double click to open colors panel to all color wells.
- Fixed model unit scaling with flowitems when creating new models.
- Fixed issues with copying network nodes in containers.
- Fixed a bug in the Excel Import on Reset not handling multiple workbooks.
- Fixed a bug in the timetables causing the duration passed in to picklists to sometimes be negative.
- Fixed issues with closing/reopening dashboard widgets.
- Fixed sizing issues with dashboard widgets.

- Fixed Rack shelf tilt.
- Fixed an issue with having TaskExecuters using navigator logic for offset travel.
- Updated Move into Highlighted object to move into the model if no object is highlighted.
- Fixed a bug with running flypaths using model run speed.
- Fixed issues with adding flowitems to User Libraries.
- Fixed sizing issues with the Crane.
- Fixed a problem where Global Variables were not being properly loaded using save/load state.
- Fixed an issue with saving views in Full Screen Mode.
- Fixes to the Conveyor's drawing.
- Fixed issues with the Shape Frame tab of the Quick Properties window.
- Fixed a bug with the Basic Conveyor causing flowitems to not always exit when they're supposed to.
- Removed FlexSim Chart and save full history.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- The order in which MTBF/MTTR triggers is fired has been changed to reflect the documentation. Now the down/up trigger fires AFTER the down/up function. This may cause problems in models that depend on the down/up triggers firing before the down/up functions.
- Excel Multi Table Import using Automatic Data Distinction will import empty cells as string data, rather than as the number 0.

FlexSim 7.0.6

- Fixed an issue in the installer regarding Optquest dll registration.
- Added support for compiling with Visual Studio and Visual Studio Express 2013.
- Fixed the duration passed into the TimeTable down function when the last row is combined with the first row.
- Fixed a bug with time tables when the model start time is on a Sunday.
- Changed the Event List to save its filters when the model is saved.
- Fixed an issue with global variables when saving or loading state files.
- Fixed a bug in Display Date and Time pick option.
- Fixed a bug with exporting bundle string data.
- Fixed an exception when documenting user commands with improperly formatted descriptions.
- Some fixes to the query() command.

FlexSim 7.0.4

- Fixed a bug with the refresh rate overlay showing for educational users.
- Fixed an issue with library installcomponents not getting installed.
- Fixed a bug with the model start time getting reset incorrectly.
- Fixed issues with Labels not showing up in Dashboard Properties window.
- Fixed issues with saving open properties windows and top level windows.
- Refactored TimeTables to minimize the number of events created.
- Fixed issues with recursive debugging hiding the debug toolbar.

FlexSim 7.0.2

- Fixed a bug with the Pick Operator with Animation picklist.
- Added scroll bar to Labels page Tree View.
- Fixed bug in Processor that showed item conveying when stopped using STATE_BLOCKED.
- Added code to read texture repeats/offsets into the assimp importer so that .ac file textures render more accurately.
- Fixed some import and display issues with SKP files.
- Fixed a bug that was occasionally causing WebKit to crash.
- Fixed a bug in the Presentation Builder that was causing the first flypoint to jump beyond the second flypoint.
- Fixed issues with the TimeTable repeating daily.
- Fixed parqty() issue on nodefunction, user commands, function_s, etc.
- Fixed bug in Dashboard to display current content of Fluid Objects.
- OnCreate now gets fired for all objects inside a container when the container is copied.
- Fixed an exception in the Startup Page's OnPreLoad when there was an invalid recent models path.
- Updated Experimenter PFM graph to draw the box plot on top of replication points.
- Made it so foreign languages' dashboard statistic names will be properly associated with the visible name that they're dragged from.
- Fixed bug with changing the model start time and it not being reflected in the model stop time.
- Fixed reset exceptions on presentation slide.
- Fixed a bug that crashes FlexSim when you call startanimation with a rank that doesn't exist.
- Fixed bug causing FlexSim to crash when copying NetworkNodes.
- Fixed round() to work properly with negative numbers and large numbers.
- Fixed triangular distribution from dividing by 0.
- Fixed issue with debugging on a script window script when the first line is commented.
- Other various bug fixes from the development list.

FlexSim 7

- New User Manual with a more usable structure.
- 64-bit version (enables FlexSim to use more RAM).
- Windowing interface overhaul to use a docked window paradigm.
- Created a Quick Properties docked window that is context sensitive. The window will display the most used properties based on the current selection or the active document window.
- Tree Find/Replace is now integrated with the Quick Properties window and has support for case-insensitive searches as well as searching for node names.
- Library Icon Grid enhancements to include filtering, collapsible groups and edit modes.
- Library Icon Grid is context sensitive and changes its display based on the current selection or the active document window.
- Added a sampler button that is placed throughout the software to allow users to sample images, 3D media, objects, nodes, numbers, strings and colors. The sampler helps to eliminate some need for writing code.
- Downloads page that gives functionality to download and install Modules, 3D Shapes, Images and Models.
- Added a Measure/Convert tool
- Improved script console allowing scripts to be saved both in individual models and to the user environment. You can also now debug your script console code.
- Improved Presentation Builder interface.
- Improved the Flowitem Bin interface including making packing modes for container flow items visible and editable. Flowitem shapes may be changed through a drag and drop from the Library Icon Grid.
- Flowitems can now have their own custom animations.
- Improved the employment of shape frames in FR objects and Flowitems.

- Added a No Select flag to all objects.
- Added a multi-table Excel export and overhauled the Excel interface to match the MTEI. The new MTEI includes an option to automatically reimport tables on reset.
- Improved Animation Creator, including dynamic animations using animation variables, more detailed editing of keyframes, and keyframe triggers. 3D shapes may be added to an animation through a drag and drop from the Library Icon Grid.
- Created a global model start date/time that is tied to TimeTables. A stop date/time may also be specified.
- Revamped TimeTable window. A daily or weekly schedule may now be imported through the MTEI.
- Added and updated several picklist popups removing all text based picklist options.
- Improvement in the Code Editor and other areas where logic is defined through draggable constructs in the library icon grid and sampler buttons throughout popups and picklist widgets to automatically add code, etc. FlexSim commands also display a short description when typing in the code editor.
- Picklist fields and many popup fields have code highlighting and autocomplete.
- Added some Flexscript implementations of lambda expressions.
- Better debugger that allows you to access the tree and other areas of FlexSim while in debug mode. Hovering over variables during debug mode will display their current value.
- Panel control GUI enhancements.
- Added dashboard constructs that will replace most need for the GUI builder: Users can now do model input through dashboards instead of having to use the GUI builder. Multiple dashboards may be created.
- You can now pick which navigator a TE is connected to through their properties page (allows you remove them from all navigators).
- New hot keys/accelerators. Ctrl+K and Ctrl+L to resize objects up or down by 5%. Ctrl+W to close the active document window or the active floating window. Updated Ctrl+Tab and Ctrl+Shift+Tab to moved between tabs in the active floating or document window.
- Added the FluidConveyor to the default fluid library.
- Can now view an object's events by right clicking an object in the 3D view and selecting View | View Object Events.
- Complete OptQuest overhaul (includes multi-core support and experimenter integration).
- Better support for importing 3D shapes. FlexSim now supports the following formats: *.wrl; *.3ds; *.dxf; *.stl; *.skp; *.dae; *.obj; *.ac; *.x; *.ase; *.ply; *.ms3d; *.cob;*.md5mesh; *.irr; *.irrmesh; *.ter; *.lxo; *.csm; *.scn; *.q3o; *.q3s; *.raw; *.off; *.mdl; *.hmp; *.scn; *.xgl; *.zgl; *.lvo; *.lvs; *.blend
- Added a new mesh class for drawing in OpenGL.
- Stereographic 3d rendering (requires workstation Quadro or FireGL card for frame-sequential rendering).
- Enhanced graphical compatibility with integrated Intel cards.
- Improved 3d rendering, including shadow rendering, specular highlights on 3ds objects, bump maps, parallax maps, etc.
- Module Development SDK, including:
    - Added SimpleDataType data type, which is a low-overhead class for fast, memory-efficient aggregation of data and for better object-oriented module code, with an easy mechanism for saving in the tree.
    - Updated visual studio wizards that work with VS 2012
    - A module sample tutorial.
    - More Documentation.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Removed 2D Shapes from objects.
- Removed the Planar view.
- Many open gl commands have been deprecated. The model update mechanism tries to replace all old usages with the new graphics usages. Usually this should work, but in some cases it may not. Note

that glBegin(), glEnd(), glVertex(), glNormal(), glTexCoord() have all been completely deprecated, and eventually will no longer work. Going forward you should use the mesh api.

- The spaceobject() command has been deprecated and no longer works. In optimizing the graphics engine we realized that a 5-20% refresh rate improvement can be attained simply by removing functionality that is solely there to make the spaceobject() command work. So we've deprecated this command. It is still in the command list so models will update, but updated models that use it will have weirdly drawn shapes.
- OnLoad is no longer dispatched by the engine when a project/tree is loaded. If you have custom objects that depend on this event, you will need to use some other mechanism to fire logic when the model loads, i.e. through the OnModelOpen, or through model libraries' OnModelOpen.
- Kinematics functionality has changed so that by default kinematics will automatically be pruned off as you pass their individual end times. Note this required a restructuring of the kinematics data, so if your models don't do it on reset anyway, you'll need to re-initialize kinematics in models that are updated from older versions.
- In previous versions, getdatastat() was documented incorrectly for the parameter p2 (degrees of freedom). It was actually interpreting p2 as the number of samples in the set, not degrees of freedom. We've fixed that by simply not using that parameter and inferring the number of samples/degrees of freedom from other parameters. This means if you used this command previously it will return different/better results in this version. Also we've changed the way the confidence interval "clamps" to percentages in order to be "safer". Again this affects the values that were returned in previous versions vs this version. See the command documentation on getdatastat() for more information.
- We changed the name of the class FlexsimObject to FlexSimObject in-line with our naming scheme going forward. We have implemented an update script that replaces all instances of "FlexsimObject" with "FlexSimObject" in updated models. This means if certain things in your model are dependent on the name "FlexsimObject" (I don't know what that could be, except for maybe dll code that uses the name FlexsimObject) there may be issues with the update.
- The assimp 3ds importer is translating some of the files (namely the robot clamps) in a way that is different than our old 3ds importer. It is possible that other 3ds files may need their offsets manually adjusted after updating.
- Fixed the per-event 50-flowitem receive limit on the queue. This will break old models if your model is dependent on this bug.
- Fixed a bug with deceleration on a network when the task executer is blocking space and is given two travel tasks in a row (this might change old models)
- Several attributes were removed, so if you use these attributes in your model you will need to update your model properly:
    - Removed Attributes: assertshape, asserttexture, billboard, distcutoff, events, instances, OnCaptured, OnCollision, OnInterrupted, state_graph, state_histo, state_percent, stats_contenthisto, stats_customgraphs, stats_throughputgraph, stats_throughputgraphmaxpoints, stats_throughputhisto, tables, textureaxis_s, textureaxis_t, travelstarttime, traveldirection, travelendtime, traveltimealpha, traveltimebeta, travelvpeak, traveldistance, travelstartx, travelstarty, travelstartz, travelendx, travelendy, travelendz, travelvmax, travelacc, traveldec
    - Removed Draw Attributes: ignoredraw, ignorezbuffer, nochildrotate, nochildscale, noondraw, nopredraw, shapetype, reflective, luminous
    - Removed Commands: travelto, traveltoupdate, ntravelto, ntraveltoupdate

FlexSim 6

- Enhanced the experimenter to use multiple cores.
- Added a new web browser GUI widget.
- Added a new Dashboard window with HTML5 canvas statistics graphs.
- Redesigned the experimenter interface to integrate the new statistics objects.
- Added new experiment variable options "number of objects in group" and "number of task executers."
- Developed web accessibility: Opening, configuring, running, and viewing models over the web (using a web browser or handheld device).
- Added a new AutoCAD dwg importer.

- Added model units and conversion windows. When building a new model, a screen will ask you what model units you will be using. When updating an old model, a screen will ask you what model units were used to build the old model. These settings are stored in the Tools folder of the model.
- Modified the picklist and trigger gui widgets to be easier to use.
- Created a new node datatype (DATATYPE_BUNDLE) for storing large amount of information efficiently (see documentation of bundle commands).
- Implemented a way to package media (3d shapes and bitmaps) into the model file so that you only need to distribute one file instead of a whole directory of files.
- Added an embedded command documentation window that can be opened by highlighting a command and pressing F1 in the code window or tree view.
- Added a FlexScript call stack to the debugger.
- Added a step-in function for FlexScript user commands and nodefunctions in the debugger.
- Added a logic builder interface for writing FlexScript logic without writing code. (You can change the default editor back to Code by unchecking 'File > Global Preferences > Environment > Use Logic Builder by Default'.)
- Enhanced the flexibility of pull logic and added a new pullitem() command for use in the Pull Strategy trigger.
- Improved templates to allow for popup gui windows on picklist options.
- Added a mechanism for multiple state profiles.
- Changed tables to always show the headers despite scrolling.
- Fixed a bug that was messing up the save operation when out of memory.
- Toggled the large-address-aware switch so that FlexSim can allocate more memory.
- Added floor() and ceil() commands.
- Added a switch for hashing the node's subnodes' names for quick lookup.
- Added an option in the Labels tab so that labels' values can be automatically reset.
- Other various fixes from the development list.

Backwards Compatibility Note: the following changes may slightly change the way updated models behave.

- Fixed a critical bug in curved network length calculations.
- Fixed issues with gettenetnode() and distancetotravel() on TEs that use "do not travel offsets and block network space."
- Made a change to distancetotravel() to base the "back-to-node" distance on the center of the object instead of the object's location.
- Changed pulling to no longer override the send-to. Now both send-to and pull must check out to transfer a flowitem.
- Changed receiveitem() so that it doesn't behave as if it were pulling.
- Made the Rack's OnEntry trigger fire before evaluating the dwell time.
- Added a new random number generator to generate seeds based on the replication number for the existing random number generator. Email support if you need a script to initialize the streams of a particular model back to the values used in prior versions.
- Added a new overload to the command tonode() to handle large memory addresses. Dlls will need to be recompiled with updated headers to obtain this fix.
- Fixed a bug where an endspeed of 0 wasn't properly telling a task executer to continue at full speed at the end of a travel task. The behavior now correctly matches the documentation.

FlexSim 5

- New undo/redo capability.
- New debugging capabilities, including:
    - An in-line, step-by-step FlexScript debugger. Just go to the code editor and click in the left margin to set a break point.

    - More strict syntax rules for FlexScript, including parameter cardinality and type checking.
    - An event log that lets you see all events that have fired in the model, filter those events, export them to csv, etc.
    - An event list that gives you a view of the list of pending events.
    - A FlexScript code profiler that lets you see what FlexScript functionality is being called, how often, and how much time is being taken up.
- A new Animation Creator that allows you to create movable sub-components of objects as well as custom animations for those sub-components.
- New .skp 3D file import capability, for using files from Google Sketchup and Google 3D Warehouse.
- Improved 3D refresh rate.
- Improved compatibility with ATI, NVIDIA, and integrated graphics cards.
- New software-based licensing using Flexnet.
- Faster and more seamless compiling process, with no creation of a new FlexSim instance.
- Fixed compiling issues for Windows Vista and Windows 7
- More detailed, navigable and configurable Experiment reporting, with additional histogram and correlation plots.
- Support for png, ico and gif texturing on 3D objects, allowing transparency without requiring the .tmp and .tpg files.
- New all-in-one code editor with tabbing and an apply button.
- Improved look on default library objects.
- New 6-axis robot (thanks Kris Geisberger).
- Improved interface for defining Conveyor layout.
- Fixed AVI Maker for Windows Vista and Windows 7.
- New and improved xml save features, allowing multiple developers to work on the same model simultaneously, as well as better integration with versioning management systems.
- Various improvements for advanced modelers and developers.
- Improved right-click menu options in the 3D View.
- Much more intuitive manipulation of objects in the 3D view.
- Users can now click on objects below the grid plane.
- Improved support for traversing window controls with the keyboard.
- Improved Find/Replace in Tree feature.
- Improved refresh rate on table views
- Fixed a crashing issue with auto-completion hints
- Fixed a bug with renaming labels
- Fixed an issue with spline points being invisible if the 3D view has a black background
- Several new FlexScript commands for string parsing.
- You can now Ctrl-Tab to switch between windows within FlexSim
- More capabilities/commands for customizing TrafficControl object logic.
- FlexSim's 3D view now has a "hover" highlight so you can see what object is under the mouse.
- More descriptive exception handling.
- Added TASKTYPE_STARTANIMATION, TASKTYPE_STOPANIMATION, and TASKTYPE_FREEOPERATORS.
- Animation of the 3D View continues while panning/zooming.
- Various minor fixes and improvements.

Getting Started

1. Getting Started
2. Interacting with FlexSim
3. Keyboard Interaction
4. Terminology
5. First Model
6. License Activation

Getting Started with FlexSim and Simulation

What is FlexSim?

FlexSim is a powerful analysis tool that helps engineers and planners make intelligent decisions in the design and operation of a system. With FlexSim, you can build a 3-dimensional computer model of a real-life system, then study that system in a shorter time frame and for less cost than with the actual system.

As a "what-if" analysis tool, FlexSim provides quantitative feedback on a number of proposed solutions to help you quickly narrow in on the optimum solution. With FlexSim's realistic graphical animation and extensive performance reports, you can identify problems and evaluate alternative solutions in a short amount of time. By using FlexSim to model a system before it is built, or to test operating policies before they are actually implemented, you will avoid many of the pitfalls that are often encountered in the startup of a new system. Improvements that previously took you months or years of trial-and-error experimentation to achieve can now be attained in a matter of days and hours using FlexSim.

Modeling

In technical terms, FlexSim is classified as a discrete-event simulation software program. This means that it is used to model systems which change state at discrete points in time as a result of specific events. Common states might be classifications such as idle, busy, blocked or down, and some examples of events would be the arrival of customer orders, product movement, and machine breakdowns. The items being processed in a discrete-event simulation model are often physical products, but they might also be customers, paperwork, drawings, tasks, phone calls, electronic messages, etc. These items proceed through a series of processing, queuing and transportation steps in what is termed a process flow. Each step of the process may require one or more resources such as a machine, a conveyor, an operator, a vehicle or a tool of some sort. Some of these resources are stationary and some are mobile; some resources are dedicated to a specific task and others must be shared across multiple tasks.

FlexSim is a versatile tool that has been used to model a variety of systems across a number of different industries. FlexSim is successfully used by small and large companies alike. Roughly half of all Fortune 500 companies are FlexSim clients, including such noted names as General Mills, Daimler Chrysler, Northrop Grumman, Discover Card, DHL, Bechtel, Bose, Michelin, FedEx, Seagate Technologies, Pratt & Whitney, TRW and NASA.

There are three basic problems which can all be solved with FlexSim:

1. Service problems – the need to process customers and their requests at the highest level of satisfaction for the lowest possible cost.
2. Manufacturing problems – the need to make the right product at the right time for the lowest possible cost.
3. Logistic problems – the need to get the right product to the right place at the right time for the lowest possible cost.

Examples of How FlexSim is Used

To give you ideas for possible projects, FlexSim has successfully been used to:

- improve equipment utilization
- reduce waiting time and queue sizes
- allocate resources efficiently
- eliminate stock-out problems
- minimize negative effects of breakdowns
- minimize negative effects of rejects and waste
- study alternative investment ideas
- determine part throughput times
- study cost reduction plans

- establish optimum batch sizes and part sequencing
- resolve material handling issues
- study effect of setup times and tool changeovers
- optimize prioritization and dispatching logic for goods and services
- train operators in overall system behavior and job related performance
- demonstrate new tool design and capabilities
- manage day-to-day operational decision making

FlexSim has been used successfully in both system design studies and in the managing of systems on a day-to-day operational basis. FlexSim has also been used for training and educational purposes. A FlexSim training model can provide insight into the complex dependencies and dynamics of a real-life system. It can help operators and management not only learn how a system operates, but learn what happens when alternative procedures are implemented. FlexSim has been used to build interactive models which can be manipulated while the model is running in order to help teach and demonstrate the cause and effects inherent in system management.

Visualization

FlexSim is a highly visible technology that can be used by forward-thinking marketers to elevate their company's image and to declare to the outside world that their company takes pride in how it operates.

It is surprising how effective an animated simulation model can be at getting management's attention and influencing their way of thinking. The animation displayed during a simulation provides a superb visual aid for demonstrating how the final system will perform.

Interacting With FlexSim

Topics

- **Creating An Object**
- **Naming An Object**
- **Editing Objects**
- **Connecting Objects**
- **View Navigation**

Creating An Object

Objects can be creating through entering a **Create Objects** mode, or by drag-and-drop:

1) Enter the **Create Objects** mode by clicking and releasing on an object in the Library window. Click again in the 3D view to create an object.

2) Alternatively, to enter the **Create Objects** mode, you may click on the button on the main toolbar. Then, click the object you wish to create in the **Library** and click again in the 3D view where you want the object to be created.

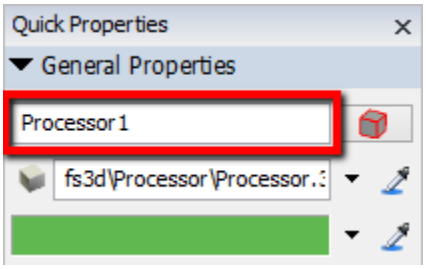
3) Click and hold the left mouse button on the object in the **Library**, then drag it to the position you want to place it in the model and release the mouse button.

Naming An Object

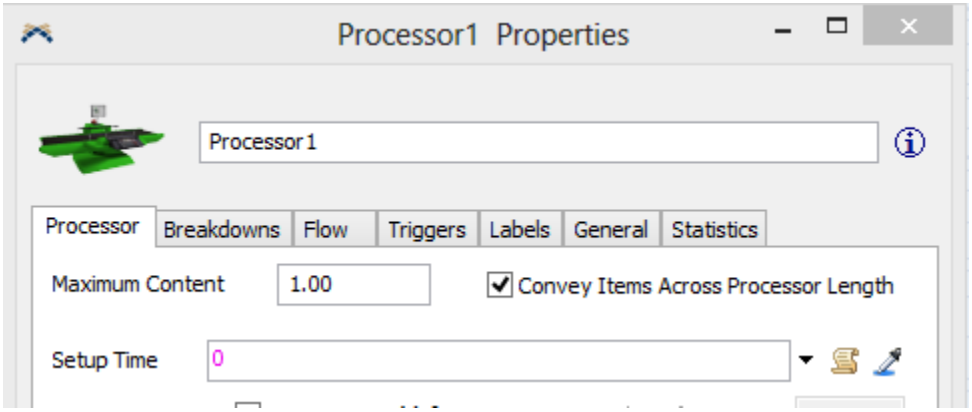
As objects are created, they will be given default names such as Source#, where # is the number of objects created since the FlexSim application was opened.

There are two methods to rename an object:

1) Click on the object in the 3D view to display the object's properties in the Quick Properties window. Then edit its name at the top of the Quick Properties window.



2) Double-click it to open its **Properties** window. Then edit its name at the top of the window and press **Apply** or **OK**.

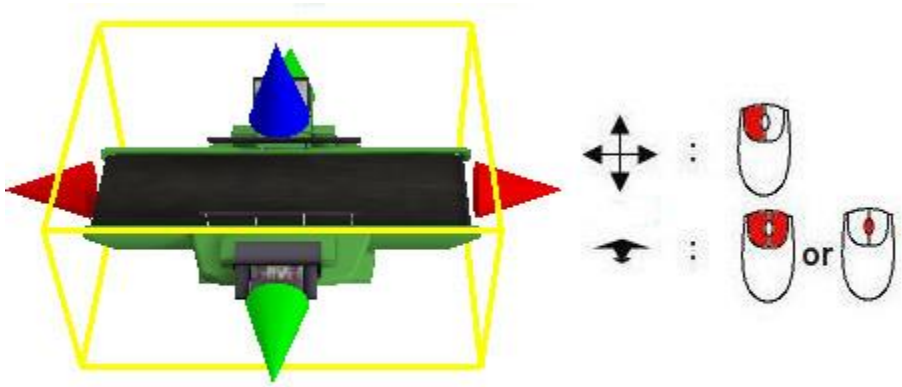


Editing Objects

Moving Objects – To move an object around in the model, click on it with the left mouse button and drag it to the position you want. You can also move the object up and down in the z direction using the mouse wheel, or by holding both the left and right mouse buttons down on the object and then dragging the mouse forward and backward.

Size and Rotation – To edit the object's size and rotation first click on the object. you should see three colored arrows along each axis of the object. To resize the object, left-click on the axis you want to resize on, and drag the mouse up or down. To edit the object's rotation, right-click on the arrow corresponding to the axis you want to rotate around, and drag the mouse forward or backward.
You may also scale an object up or down by 5% by holding the Ctrl key and pressing the K or L key.

Note: You can toggle Resizing and Rotating Objects to be on or off by selecting the main menu option **Edit > Resize and Rotate Objects**.



Properties – All FlexSim objects have a number of pages or tabs that present variables and information that the modeler can change based on the requirements of the model. See here for more details.

Destroying Objects – To destroy an object, click on that object and press the Delete key.

Connecting Objects

Ports are created and connected in one of two ways:

1) By clicking on one object and dragging to a second object while holding down different letters on the keyboard. If the letter "A" is held down while clicking-and-dragging, an **output port** will be created on the first object and an **input port** will be created on the second object. These two new ports will then be automatically connected. Holding down the "S" key will create a **central port** on both objects and connect the two new ports. Connections are broken and ports deleted by holding down the "Q" for input and output ports and the "W" key for central ports. The following table shows the keyboard letters used to make and break the two types of port connections:

	Output - Input	Center
Disconnect	Q	W
Connect	A	S

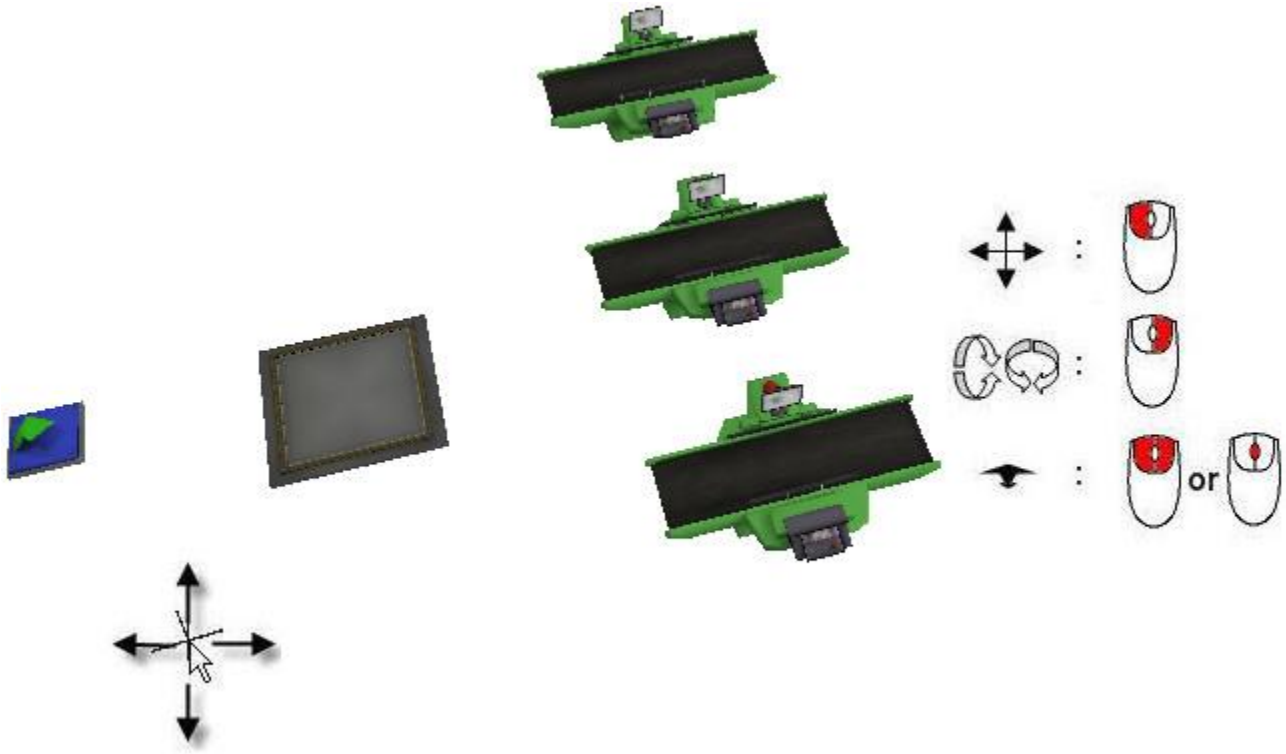
2) By entering the **Connection Mode**, which can be entered by clicking the button in the main toolbar. Once in the Connection Mode, there are a couple of ways to make a connection between two

objects. You can either click on one object, then click on another object, or you can click and drag from one object to the next as with method one. Either way, keep in mind that the flow direction of a connection is dependent on the order in which you make the connection. Flow goes from the first object to the second object.

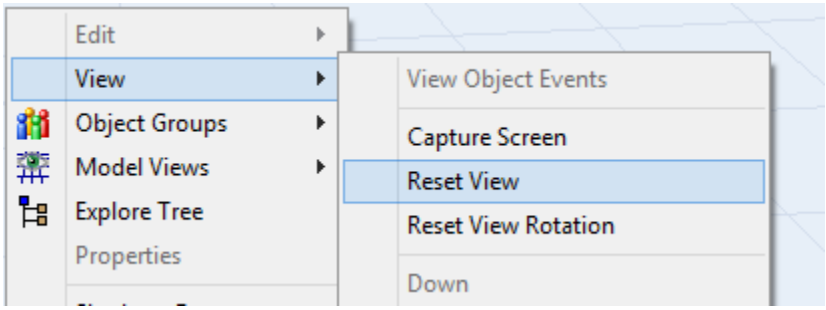
Connections can be broken by clicking the button then clicking or dragging from one object to another in the same manner as when you connected them. Center port connections are not affected by the order in which the objects are connected.

View Navigation

Basic Navigation – To move the model view point, click in an empty area of the view with the left mouse button, and drag the mouse around. To rotate the model view point, click in a blank area with the right mouse button and drag the mouse around. To zoom out or in, use the mouse wheel or hold both left and right mouse buttons down and drag the mouse.

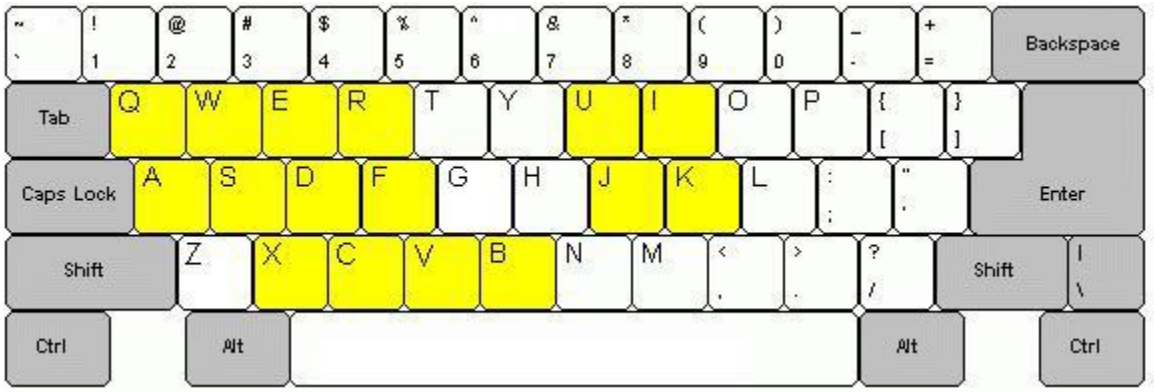


Reset View – You can reset the view to the default view point by right clicking in empty grid space, and select the menu **View > Reset View**.



Keyboard Interaction

When you are working in the 3D view, you will use several keys on the keyboard to build, customize, and get information from the model. The figure below shows the keyboard layout. Keys that are highlighted in yellow have meaning when interacting with Flexsim.



A, J: context sensitive connect

The A key is used to connect two objects depending on the type of objects. Hold down the A key, click on one object, holding the mouse button down, and drag to another object, releasing the mouse button on that object. Usually this connects the output ports of one object to the input ports of another object. For NetworkNodes, however, the A key connects a NetworkNode to TaskExecuters as travellers, to FixedResources as travel gateways, and to other NetworkNodes as travel paths. You can also use the J key if you are left handed. If you connect two objects with the A key, and don't see any changes, first, click on an empty area in the 3D view and make sure the Show Connections button is checked in the Quick Properties window. If still no change is apparent, then those objects are probably not supposed to be connected with the A key.

Q, U: context sensitive disconnect

The Q key is used to disconnect two objects depending on the type of objects. Hold down the Q key, click on one object, holding the mouse button down, and drag to another object, releasing the mouse button on that object. Usually this disconnects the output ports of one object from the input ports of another object. For NetworkNodes, however, the Q key disconnects a NetworkNode from TaskExecuters as travellers, from FixedResources as travel gateways, and sets one-way of a travel path connection to "no connection" (red). You can also use the U key if you are left handed.

S, K: central port connect

The S key is used to connect central ports of two objects. Central ports are used for referencing purposes, using the centerobject() command. Hold down the S key, click on one object, holding the mouse button down, and drag to another object, releasing the mouse button on that object. You can also use the K key if you are left handed.

W, I: central port disconnect

The W key is used to disconnect central ports of two objects. Hold down the W key, click on one object, holding the mouse button down, and drag to another object, releasing the mouse button on that object. You can also use the I key if you are left handed.

D: context sensitive connect

The D key is a second key for context sensitive connecting. The NetworkNode and the TrafficControl both implement this connection.

E: context sensitive disconnect

The E key is a second key for context sensitive disconnecting. The NetworkNode implements this connection.

X: context sensitive click/toggle

The X key is used to change an object or view information on the object, dependent on the type of object. Hold the X key down, and click on the object. The NetworkNode will toggle the whole network through different viewing modes. The X key also creates new spline points on a network path. Racks will also toggle through different viewing modes. A conveyor will reposition downstream conveyors to be flush with conveyor end points.

B: context sensitive click/toggle

The B key is an additional key used to change an object or view information on the object, dependent on the type of object. Hold the B key down, and click on the object. The NetworkNode will toggle the whole network through different viewing modes. The TrafficControl also uses the B key.

V: view input/output port connections

The V key is used to view an object's input/output port connections. Hold the V key down, and click on an object, holding both the V key and the mouse button down. If the mouse button is released first, then the information will disappear, but if the V key is released first, the information will persist.

C: view central port connections

The C key is used to view an object's central port connections. Hold the C key down, and click on an object, holding both the V key and the mouse button down. If the mouse button is released first, then the information will disappear, but if the C key is released first, the information will persist.

F: create library objects

The F key is used to quickly create library objects. Select an object in the library icon grid by clicking on it. Then click in the ortho/perspective view, and press and hold the F key down. Then click in the ortho view in the location you would like to create the object. The object will be created at that position.

R: create and connect library objects

The R key is like the F key, except it also connects consecutively created objects with an A connection.

Hot Keys / Accelerators

Ctrl + Space - Start and stop the model run.

Ctrl + Left Arrow - Reset the model.

Ctrl + Right Arrow - Step to the next model event.

Ctrl + Up Arrow - Increase the simulation run speed.

Ctrl + Down Arrow - Decrease the simulation run speed.

Ctrl + F - Find text in the open view.

Ctrl + H - Find and replace text in the open view.

Ctrl + C - Copy the selected object(s) to the clipboard.

Ctrl + X - Cut the selected object(s) to the clipboard.

Ctrl + V - Paste the object(s) in the clipboard.

Ctrl + T - Open a new tabbed document window (based on the currently active document window, 3D or Tree only)

Ctrl + Tab - Switch to the next window tab.

Ctrl + Shift + Tab - Switch to the previous window tab.

Ctrl + L - Scale the selected object(s) up by 5%.

Ctrl + K - Scale the selected object(s) down by 5%.

Ctrl + Shift + D - Add a keyframe to the presentation builder.

Ctrl + W - Close the active document window or floating window.

F1 - Open the Command Helper and search for the selected text.

P - Takes a screen shot of the active view and saves it to projects folder/screenshots (adds pointer data if highlighted a node in the tree)

Tree Window Shortcuts

The following are available in the Tree Window:

Spacebar - Insert a new node after.

Enter - Insert a new node into.

Del - Delete selected node(s).

N - Add number data to the highlighted node.

T - Add string (text) data to the highlighted node.

O - Add object data to the highlighted node.

P - Add pointer data to the highlighted node.

Shift + Del - Delete node data.

FlexSim Terminology

Before you start your first model it will be helpful to understand some of the basic terminology of the software.

FlexSim Objects

FlexSim objects simulate different types of resources in the simulation. An example is the Queue object, which acts as a storage or buffer area. The Queue can represent a line of people, a queue of idle processes on a CPU, a storage area on the floor of a factory, a queue of waiting calls at a customer service center, etc. Another example of a FlexSim object is the Processor object, which simulates a delay or processing time. This object can represent a machine in a factory, a bank teller servicing a customer, a mail employee sorting packages, an epoxy curing time, etc.

FlexSim objects are found in the Library Icon Grid.

Flowitems

Flowitems are the objects that move through your model. Flowitems can represent parts, pallets, assemblies, paper, containers, telephone calls, orders, or anything that moves through the process you are simulating. Flowitems can have processes performed on them and can be carried through the model by material handling resources. In FlexSim, flowitems are generated by a Source object. Once flowitems have passed through the model, they are sent to a Sink object.

Flowitems are managed in the Flowitem Bin.

Labels

Labels are strings or numbers that are stored on Flowitems and objects. Labels can be dynamically altered through the course of a process flow. Labels can be useful for storing information like cost, processing time and other information. Labels can be accessed through the object's quick properties or its Labels Page.

Click here to learn more about labels.

Itemtype

The itemtype is a special label that is placed on the flowitem that could represent a barcode number, product type, or part number. FlexSim is set up to use the itemtype as a reference in routing flowitems.

Click here to learn more about itemtypes.

Ports

Every FlexSim object has an unlimited number of ports through which they communicate with other objects. There are three types of ports: input, output, and center.

- **Input and Output Ports:** These ports are used in the routing of flowitems. For example, a mail sorter places packages on one of several conveyors depending on the destination of the package. To simulate this in FlexSim, you would connect the output ports of a Processor object to the input ports of several Conveyor objects, meaning once the Processor (or mail sorter) has finished processing the flowitem (or package), it sends it to a specific conveyor through one of its output ports.
- **Center Ports:** These ports are used to create references from one object to another. A common use for central ports is for referencing TaskExecuter objects such as Operators , Fork Lifts , and Cranes from FixedResources such as Processors , Queues , or Conveyors .

Click here to learn more about ports.

First Model

Description

In this model we will look at the process of manufacturing three types of products in a factory. In our simulation model, we will associate an itemtype value with each of the three product types. These three types all arrive intermittently from another part of the factory. There are also three machines in our model. Each machine can process a specific product type. Once products are finished at their respective machines, all three types of products must be tested at a single shared testing station for correctness. If they have been manufactured correctly, they are sent on to another part of the facility, leaving our simulation model. If they were manufactured incorrectly, they must return to the start of the simulation model to be re-processed by their respective machines. The goal of the simulation is to find where the bottleneck is. Is the testing machine causing the three other machines to back up, or is it being starved because the three machines can't keep up with it? Is the amount of buffer space before the tester important?

Applying the Model to Different Industries

While we are using the manufacturing industry for this example, the same simulation model can be applied to other industries. Take a copy shop for example. A copy shop has three main services: black and white copies, color copies, and binding. During business hours, there are three employees working. One employee handles black and white copy jobs, another handles color copy jobs, and the third handles binding jobs. There is also a cashier to ring up finished orders. Each customer that enters the copy shop gives a job to the employee that specializes in his type of job. As each job is finished, it is placed in a queue for the cashier to finalize the sale and give to the customer. However, sometimes the customer is not satisfied with the job that was done. In such cases, the job must be given back to the appropriate employee to be done again. This scenario represents the same simulation model as the one described above for the manufacturing industry. Here, though, you may be more concerned with the customer queue and the time they spend waiting, as slow service can be very costly to a copy shop's business.

Here's another example of the same simulation model applied to the transportation industry. Commercial shipping trucks traveling over a bridge from Canada into America must go through a customs facility before being allowed to enter the country. Each truck driver must first get the proper paperwork necessary, and then pass through a final inspection of the truck. There are three general categories of trucks. Each category has a different type of paperwork to fill out and must apply at a different department of the customs facility. Once paperwork is finished, all categories of trucks must go through the same inspection process. If they fail the inspection, then they must go through more paperwork, etc. Again, this situation contains the exact same simulation elements as the manufacturing example, only applied to the transportation industry. Here, you may be interested in how far the trucks back up across the bridge. If they back up for miles and thus block traffic into the neighboring Canadian city, then you may need to change how the facility operates.

Building the Model

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials.

Steps

- **Step 1: Start FlexSim**
- **Step 2: Select Units**
- **Step 3: Create the Objects**
- **Step 4: Connect the Objects**
- **Step 5: Define the Inter-Arrival Time**
- **Step 6: Assign an Itemtype and a Color**
- **Step 7: Define the Queue's Maximum Content**
- **Step 8: Define Queue1's Routing**

- **Step 9: Define Process Times**
- **Step 10: Define Queue2's Maximum Content**
- **Step 11: Define Tester's Process Time**
- **Step 12: Define Tester's Routing**
- **Step 13: Reset and Run the Model**
- **Creating a Dashboard**
- **Randomness**
- **Results**

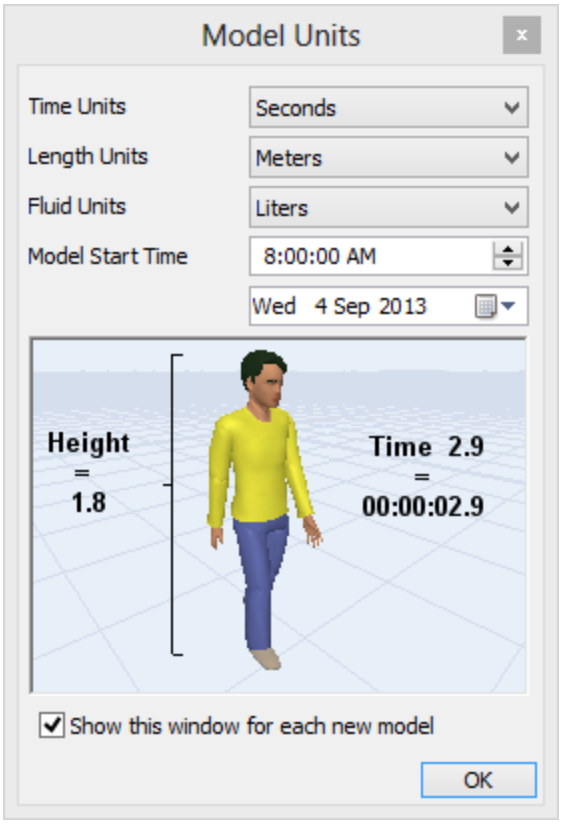
Step 1: Start FlexSim

- Open FlexSim by double-clicking on the FlexSim icon on your desktop. The **Start Page** will appear. Select the "New Model" option in the upper left hand corner of the window.

Step 2: Select Units

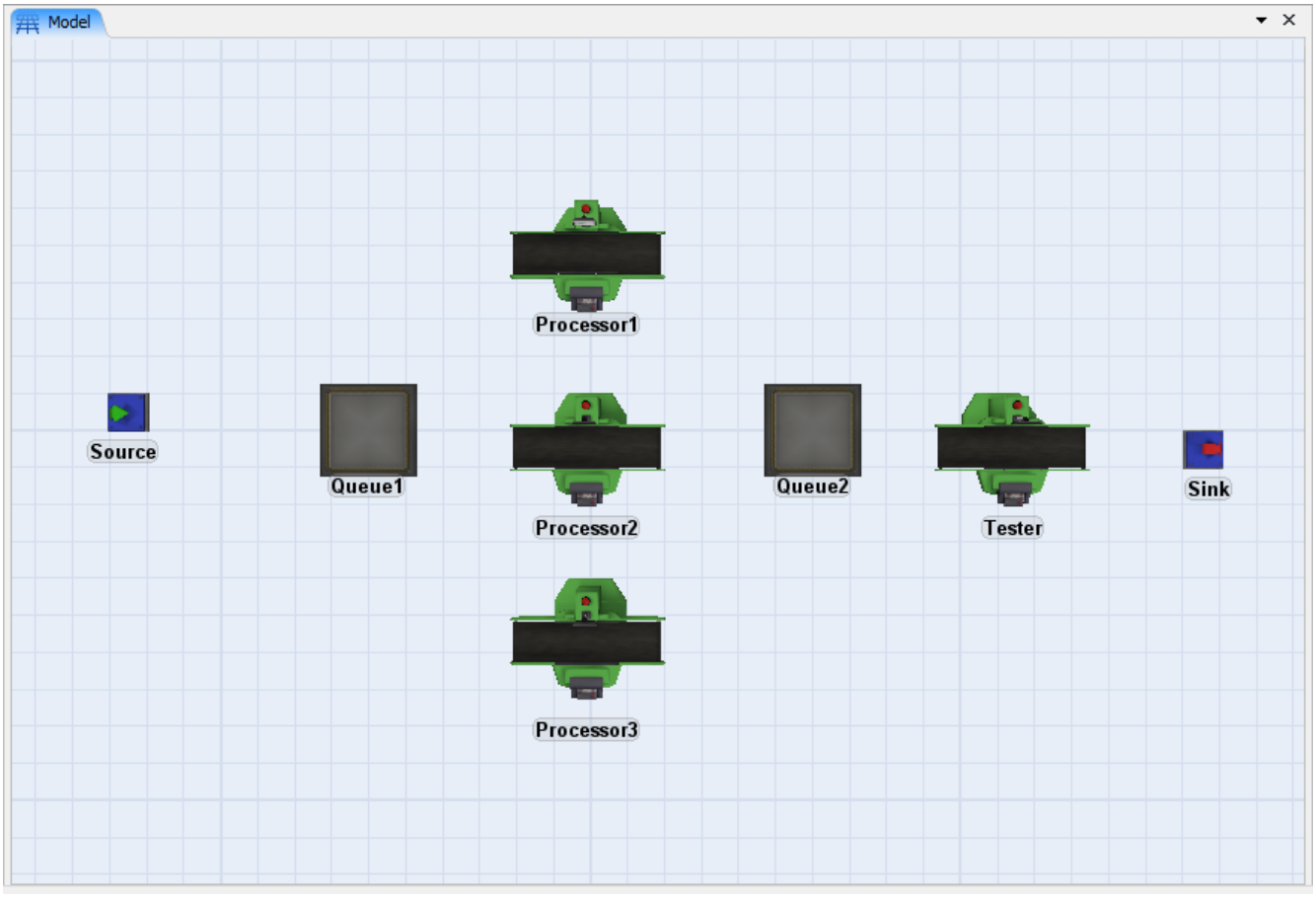
FlexSim allows the user to select appropriate units for a model. By default the **Model Units window** will appear for each new model. You can select units for time, length, fluids and a Model Start Time. The units you choose will be used throughout the model. The Model Start Time may be changed after the model is created, however, the Time, Length and Fluid units CANNOT be changed. For this model, use the following:

- **Time Units:** Seconds.
- **Length Units:** Meters.
- **Fluid Units:** Liters.
- **Model Start Time:** Leave as default.



Step 3: Create the Objects

- Create a Source, two Queues, four Processors, and a Sink in the model. Name and place them as shown below (note that one of the Processor objects will be the "Tester").
- To review the process for creating objects in FlexSim, refer to the Creating An Object section of the Interacting with FlexSim page. To review how to rename an object, refer to the Naming An Object section.

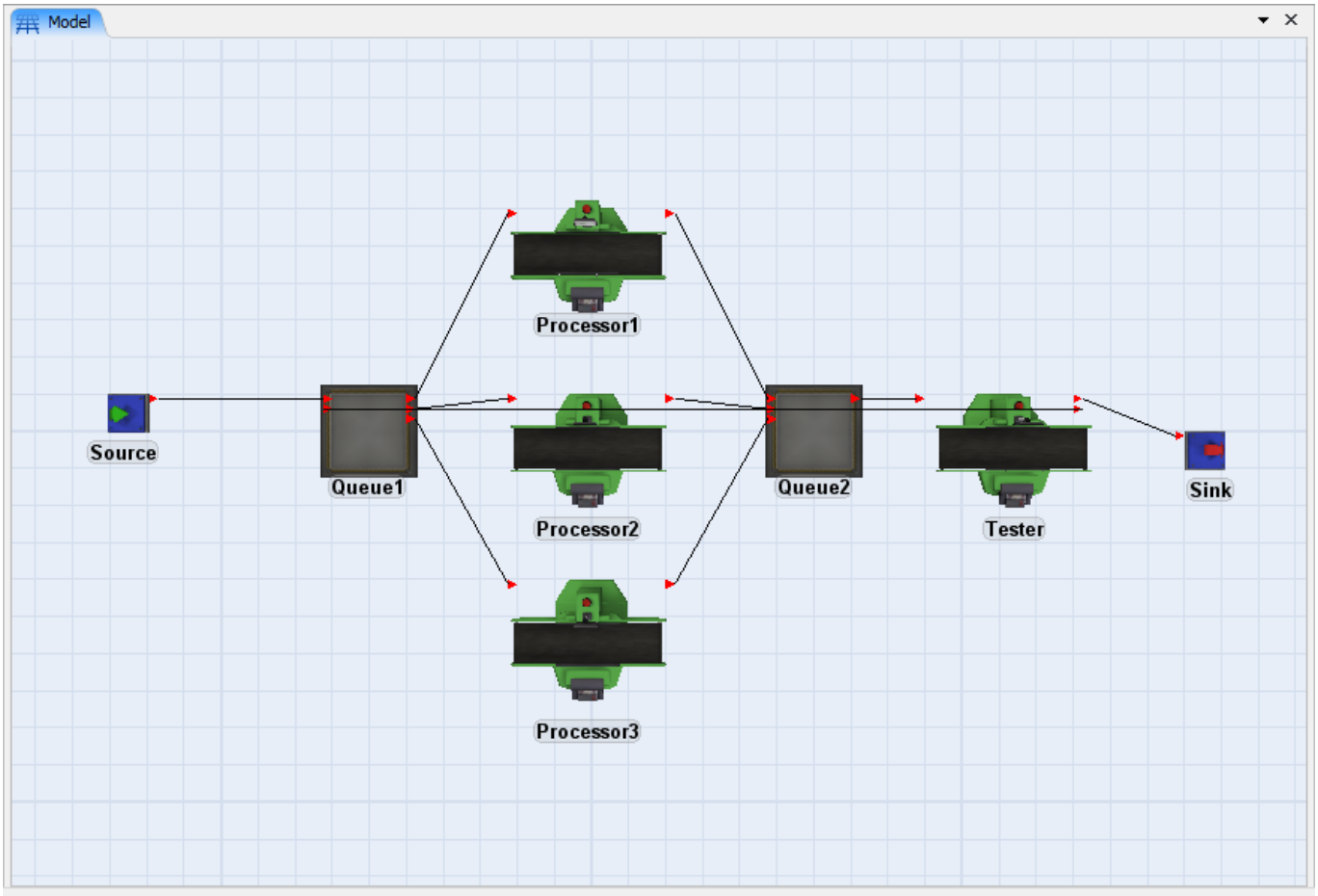


Step 4: Connect the Objects

To review how to connect objects, refer to the Connecting Objects section of the Interacting with FlexSim page.

Notice the Tester object has an output connection to Queue1. This will allow rejected items to be sent back to the start of the process.

- Connect *Source* to *Queue1.*
- Connect *Queue1* to *Processor1*, *Processor2*, and *Processor3.*
- Connect *Processor1*, *Processor2*, and *Processor3* to *Queue2.*
- Connect *Queue2* to *Tester.*
- Connect *Tester* to *Sink* and *Queue1.*



The next step is to change the properties of the different objects so they will behave as specified in the model description. We will start with the source and work our way to the sink.

Each object has its own properties window through which data and logic are added to the model. Double-clicking on an object accesses the object's properties window.

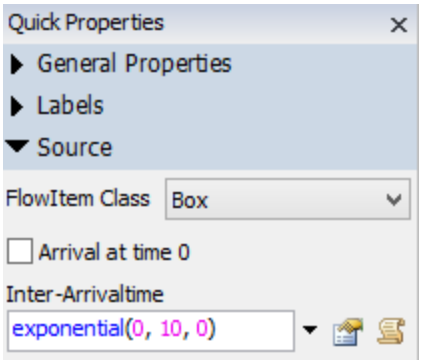
For this model, we want three different product types to enter the system. To do this, each flowitem's itemtype will be assigned an integer value between one and three using a uniform distribution (see FlexSim Concepts for more information about itemtypes). This will be accomplished using the source's exit trigger.

Step 5: Define the Inter-Arrival Time

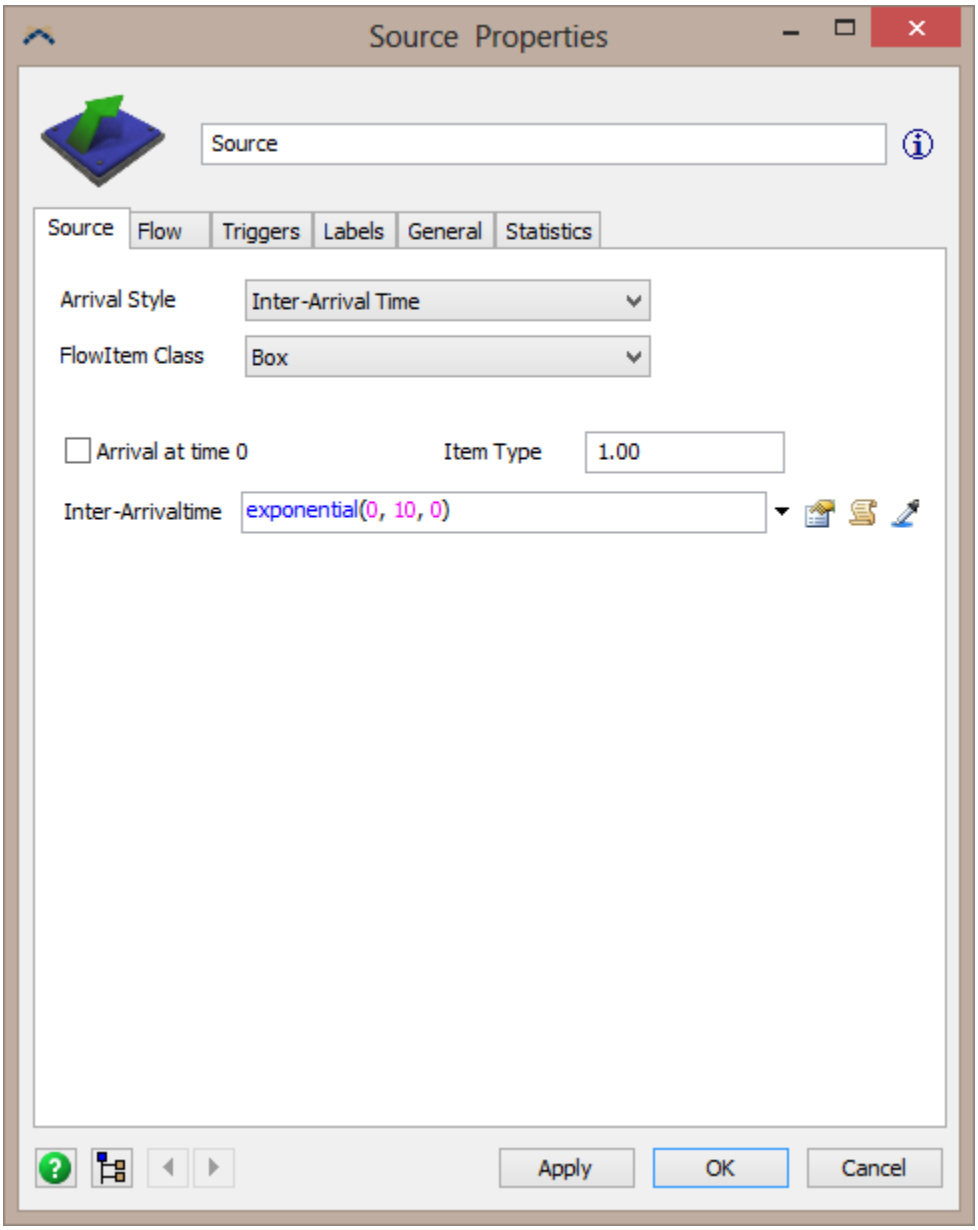
Products arrive every 5 seconds, exponentially distributed. The Source, by default, uses an exponentially distributed inter-arrival time, but you will change the mean of that distribution. Statistical distributions like exponential distribution are used throughout simulation in order to model the variations that occur in real-life systems.

You may edit the Source's Inter-Arrival Time from two different windows:

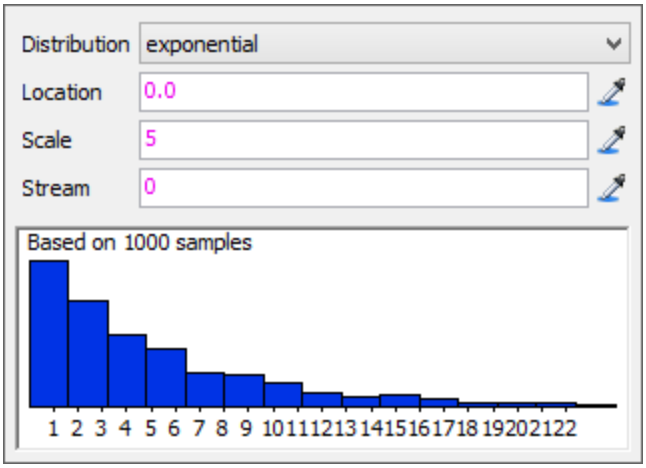
- 1) Click on the *Source* to bring up its properties in the Quick Properties window.



- 2) Double-click on the *Source* to bring up its **Properties** window.



- On the **Source** tab, click on the button. A popup will appear.
    - Set **Distribution** to exponential.
    - Set **Location** to 0.
    - Set **Scale** to 5.
    - Set **Stream** to 0.



Click anywhere outside the popup to save these settings.

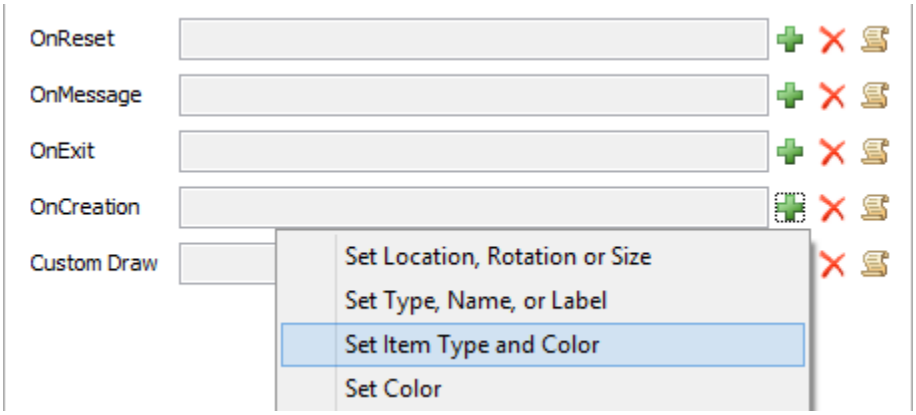
Remember that units were set at the beginning. Setting **Scale** to 5 sets the mean of the distribution to 5 seconds. If the units had been set to hours, the mean would have been 5 hours.

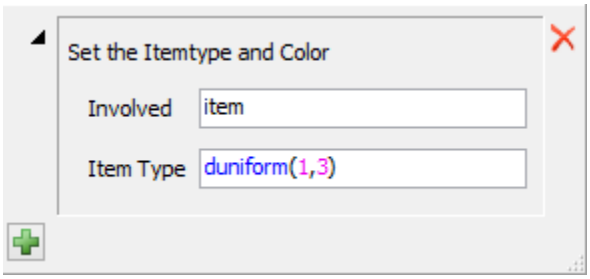
- If you edited the Inter-Arrival time through the Quick Properties window, you'll need to open the Source's **Properties** window in order to perform Step 6. This can be done by clicking the **More Properties** button under the General Properties section of the Quick Properties.

Step 6: Assign an Itemtype and a Color

The next thing we need to do is assign an itemtype number to the flowitems as they enter the system. This value is uniformly distributed between 1 and 3, meaning the chance that the entering product is type 1 is just as likely as it is type 2, which is just as likely as it is type 3. The best way to do this would be to change the itemtype in the OnCreation trigger of the Source.

- Click the **Triggers** tab. Add a function (press the button) to the **OnCreation** trigger. Select **Set Item Type and Color** from the list. A popup will appear.



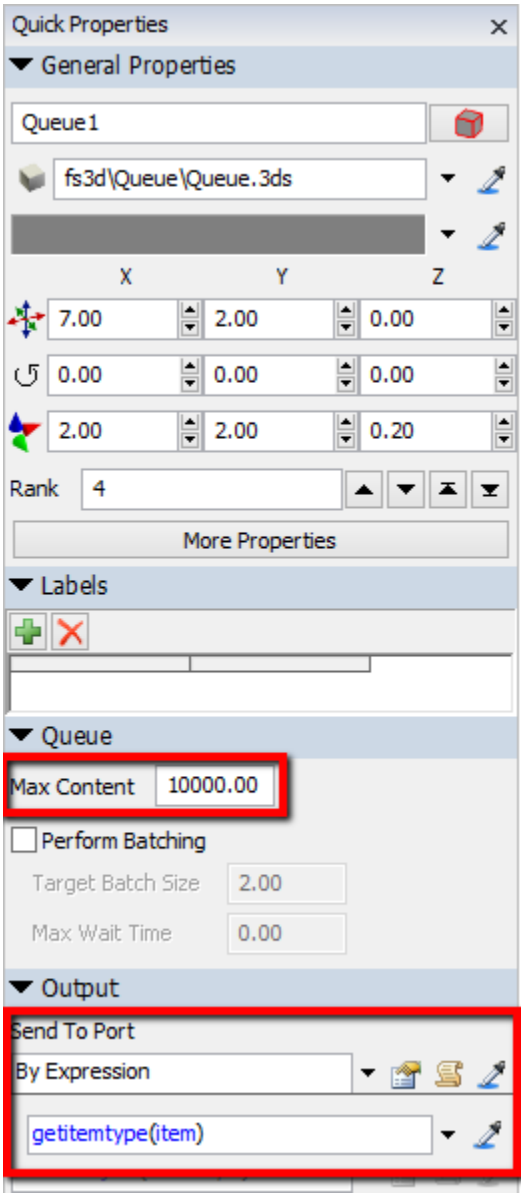


The duniform distribution is similar to a uniform distribution except that instead of returning a real number it will only return whole numbers. Click **OK** to apply the changes and close the window.

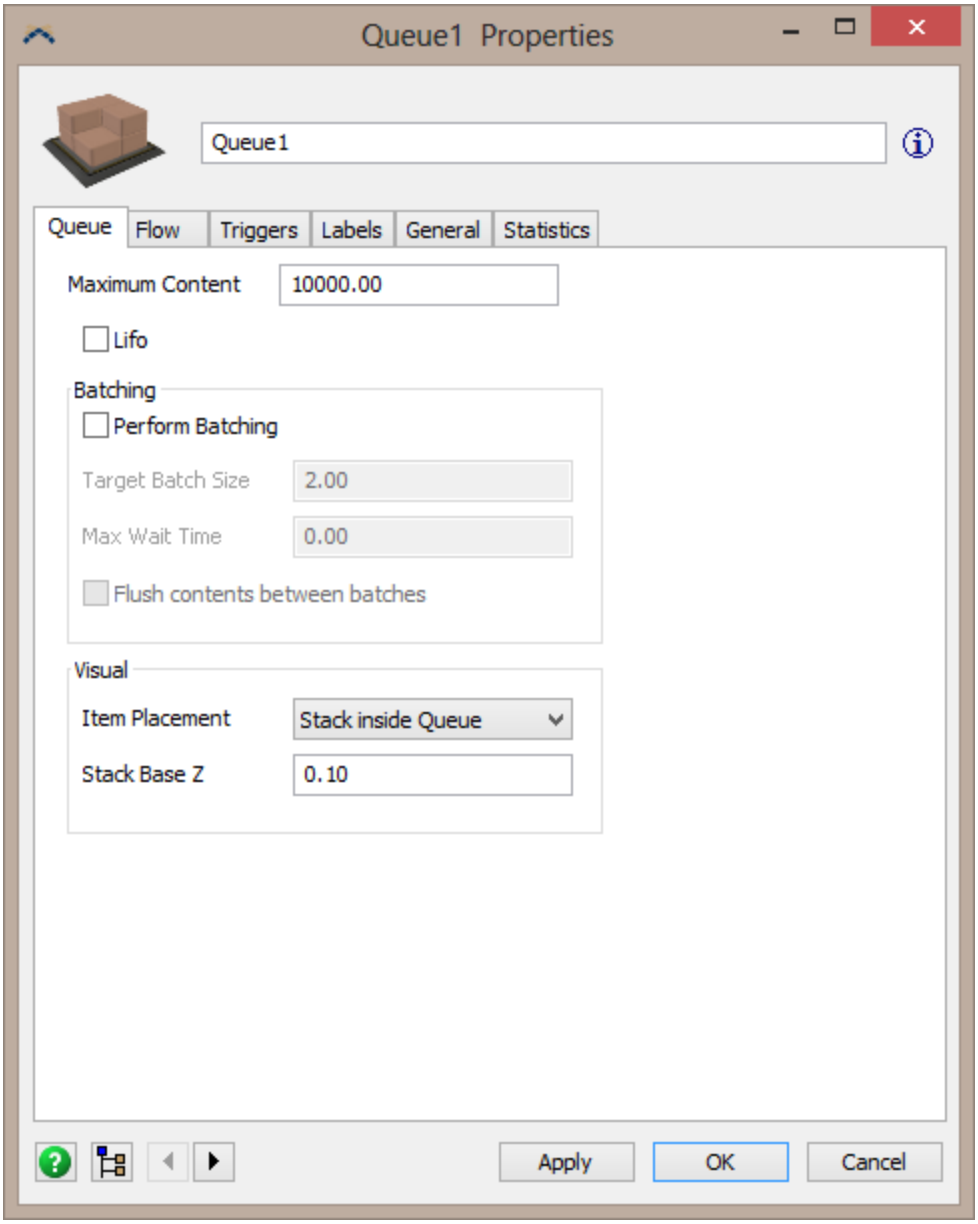
Step 7: Define the Queue's Maximum Content

The next step is to edit the Queue. There are two things we need to configure on *Queue1*. First we need to set the Maximum Content of the Queue. Second, we need to have the Queue send itemtype 1 to *Processor1*, itemtype 2 to *Processor2*, and itemtype 3 to *Processor3*.

This step, along with step 8 can be done through the Quick Properties window as shown below, or by opening the Queue's Properties window as described.

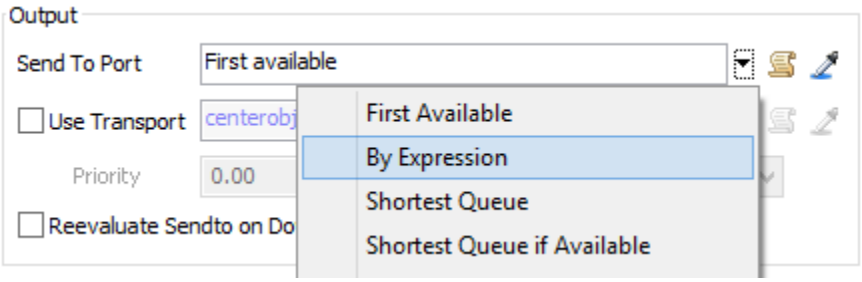


- Double-click on *Queue1* to open its **Properties** window.
- On the **Queue** tab, change the **Maximum Content** to 10000.
- Click **Apply**, but do not close the Properties window.

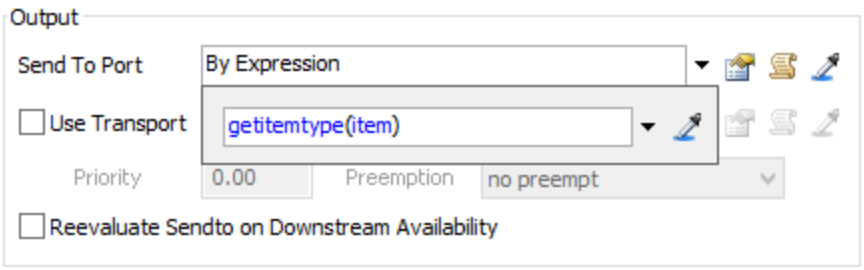


Step 8: Define Queue1's Routing

- Click the **Flow** tab.
- Under **Output**, select **By Expression** from the **Send To Port** drop-down list.



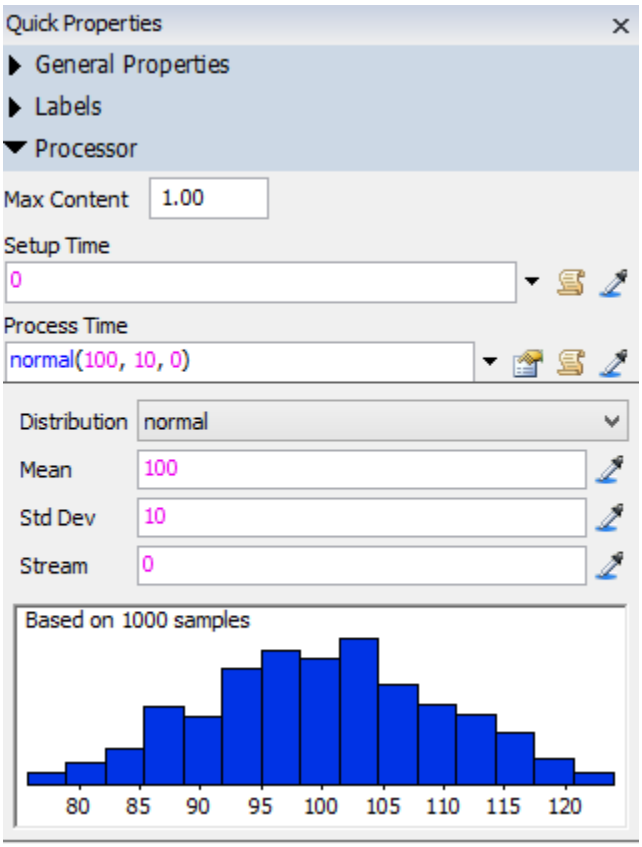
- A popup with suggested expressions will appear. The default expression for **By Expression** is **getitemtype(item)** . This will send type 1 to port 1, type 2 to port 2, and so on. Click anywhere outside popup to close it, and then click **OK** to apply the changes and close the window.



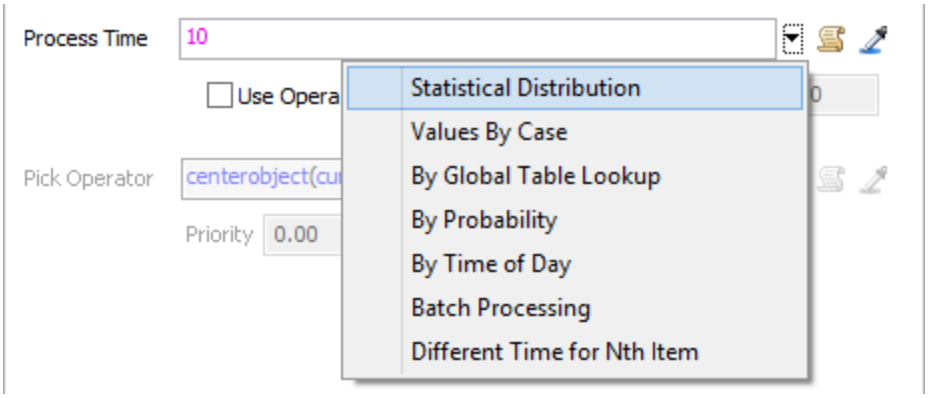
Step 9: Define Process Times

The next step is to set the processing times for the three processors.

As described for Step 7, the **Process Time** can be set through the Quick Properties window by clicking on the object once in the 3D view.



- Double-click on *Processor1* to open its **Properties** window.
- On the **Processor** tab, select **Statictical Distribution** from the **Process Time** list.
- In the **Statictical Distribution** popup, set **Distribution** to **exponential.** Use the default parameters given for this distribution.
- Click **OK** to apply the changes and close the window.
- Repeat this step for *Processor2* and *Processor3*.



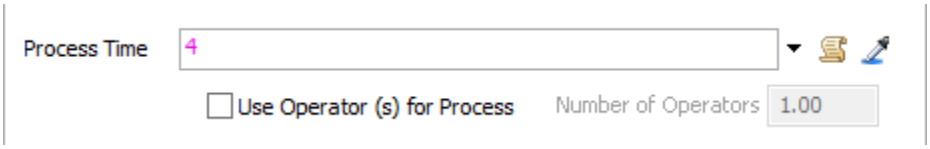
Step 10: Define Queue2's Maximum Content

Follow Step 7 to change Queue2's **Maximum Content** to 10000.

Step 11: Define Tester's Process Time

As described in Step 9, this can be set through the Quick Properties window as well.

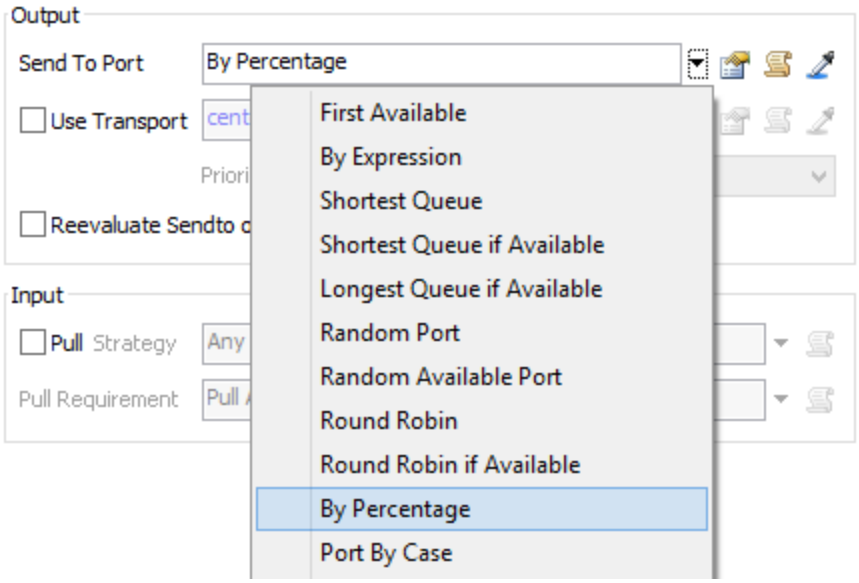
- Double-click on *Tester* to open its **Properties** window.
- On the **Processor** tab, highlight all the text in the **Process Time** field.
- Replace the text with **4**. This sets the process time to a constant four seconds.
- Click **Apply**, but do not close the Properties window.



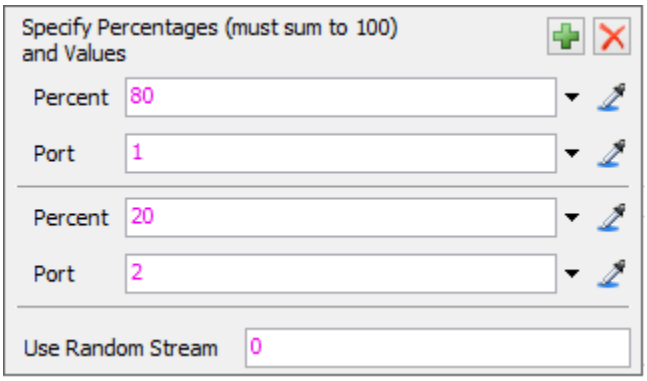
Step 12: Define Tester's Routing

Now we need to configure the testing station to send bad products back to the beginning of the model, and to send good products to the sink. When you created this object's connections, you should have first connected it to the sink, then connected it back to the first queue. This ordering will have made the first output port of the testing station be connected to the sink and the second output port be connected to *Queue1*. You can verify that the ports are correct by clicking Output Ports in the **Ports** panel, which is at the bottom of the **General** tab. If the ports are out of order, you can use the "Rank ^" and "Rank v" buttons to reorder the ports. Now we want to route to the appropriate port number based on a certain percentage.

- Click the **Flow** tab**.** Select **By Percentage** from the **Send To Port** list.



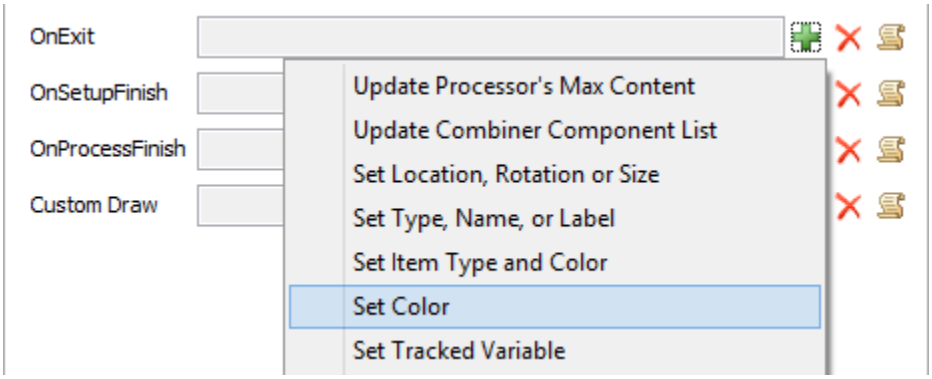
- Use the ✚ to add another field.
- Fill the fields to match the picture below.

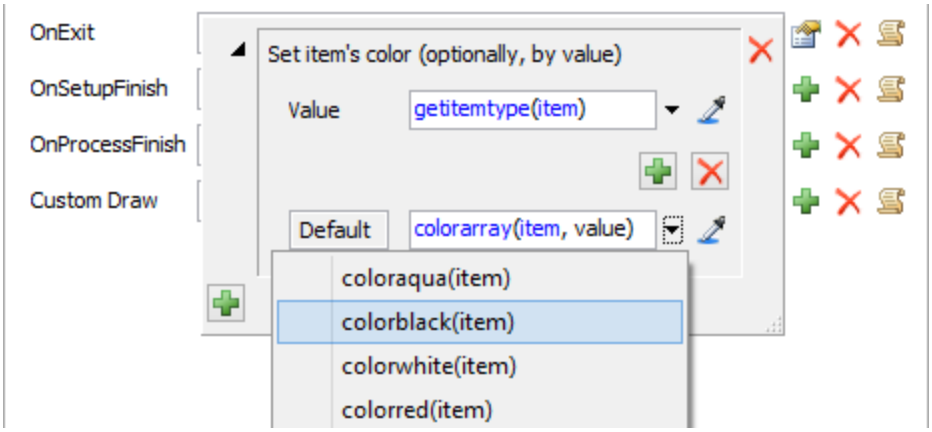


This means that 80 percent of the products (the correctly manufactured products) will be sent through output port 1 to the Sink, and 20 percent (the incorrectly manufactured products) will be sent through output port 2 back to the first queue.

One more thing we might want to do is visually distinguish items that have already been through the testing station and have been sent back to the first queue.

- Click the **Triggers** tab. Add a function (click the ✚ button) to the **OnExit** trigger and select the **Set Color** option. Select colorblack(item) from the list.



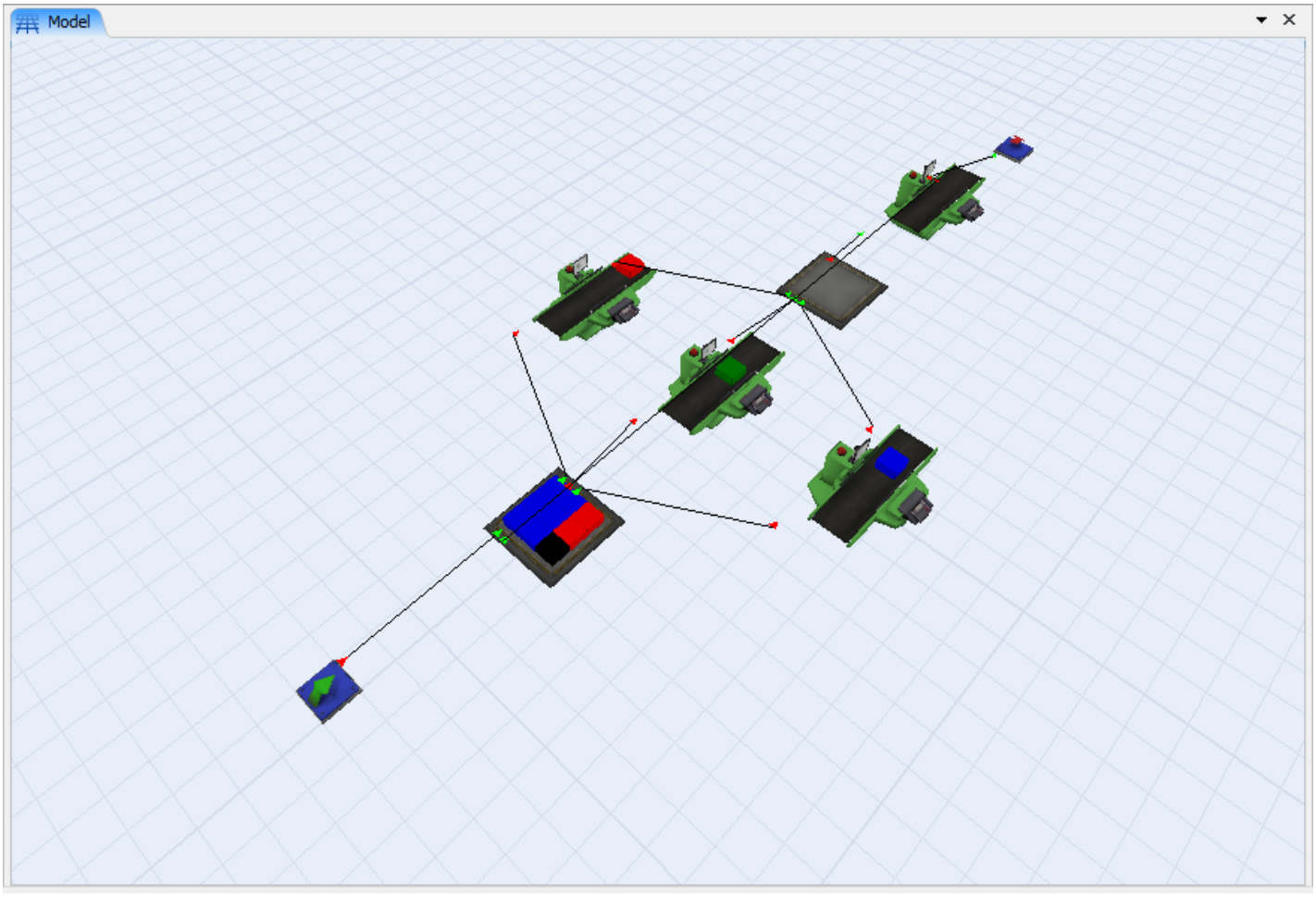


- Press **OK** to close the Properties window.

Step 13: Reset and Run the Model

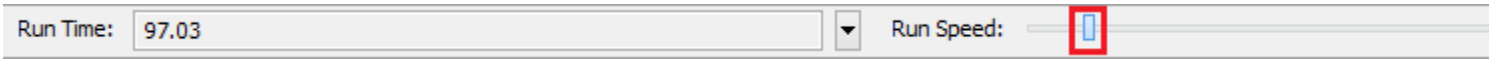
- Click on the Reset button, located at the upper left-hand corner. Resetting the model sets all system variables to their starting values and clears any flowitems present in the model. Resetting is also necessary any time new connections are made between objects.
- Click the Run button, located right next to the reset button.

The model should now start to run. Flowitems should move from the first queue, into one of the three processors, then to the second queue, into the testing station, and from there to the sink, with some being re-routed back to the first queue. Re-routed items will be colored black.



To stop the model, press the Stop button at any time. Later you will learn how to run a model for a specified time, and for a specified number of iterations. Running a model more than once is important when statistical distributions have been used in the model definition.

To speed the model up or slow it down, move the Simulation time slide bar at the top of the window to the right or left. Alternatively, you can press the Ctrl + Down Arrow and the Ctrl + Up Arrow to increase or decrease the run speed.



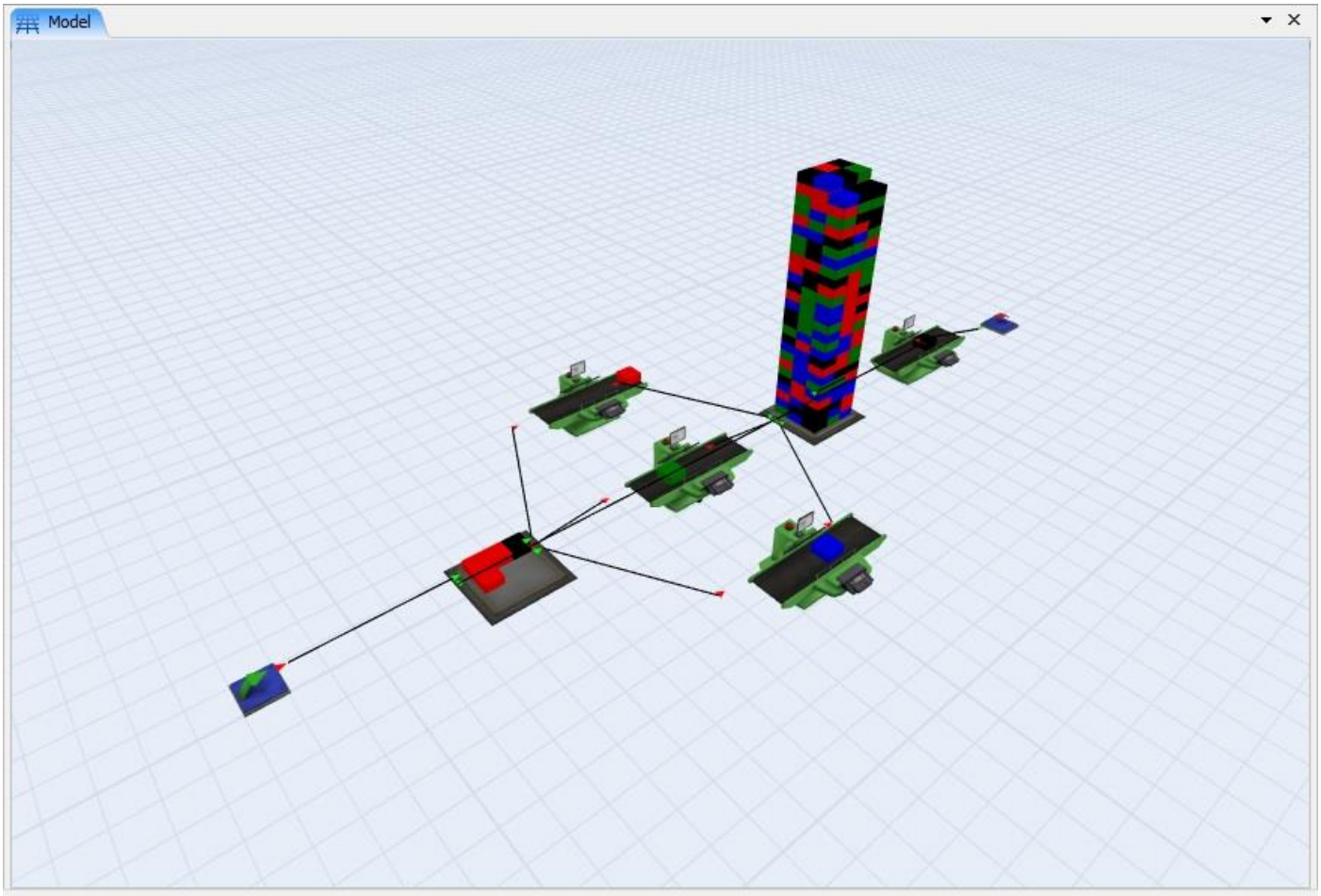
Moving the slide bar changes how fast the simulation time proceeds relative to real time. It has no effect on model results.

We have now completed building the model. Let's look at some of the statistics the model generates.

Creating a Dashboard

Finding the Bottleneck

In the model description, we said that we wanted to know where the bottleneck was in the system. There are several ways to determine this. First, you can simply examine the visual size of each queue. If one queue in the model consistently has many products backed up in it, then that is a good indication that the processing station(s) that it feeds are causing a bottleneck in the system. In running this model, you'll notice that the second queue very often has a lot of products waiting to be processed, whereas the first queue's content is usually 20 or less, as shown below.

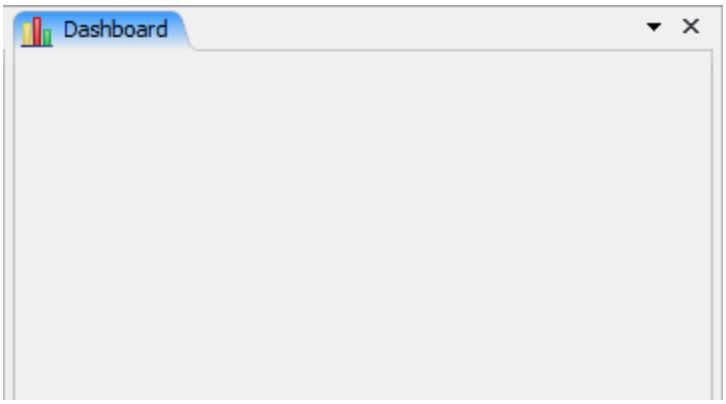


Another way of finding the location of a bottleneck is by examining the state statistics of each of the processors. If the three upstream processors are always busy, while the testing station is often idle, then the bottleneck is likely to be at the three upstream processors. On the other hand, if the testing station is always busy, while the upstream processors are often idle, then the bottleneck is probably at the testing station.

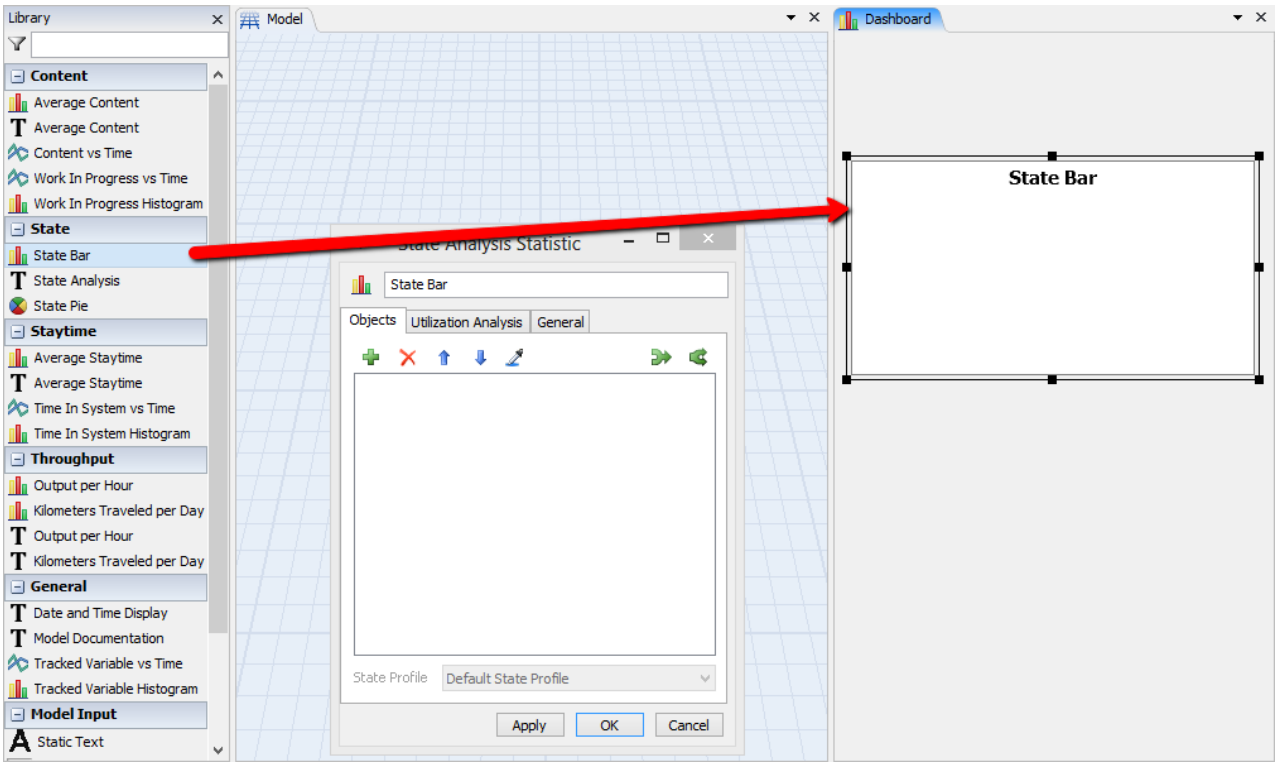
Evaluating the New Configuration

Run the model for at least 50,000 seconds. Notice first that *Queue2* is now almost always empty, whereas the Queue for the 3 processors backs up quite often. Let's use the dashboard to compare the two testers side by side.

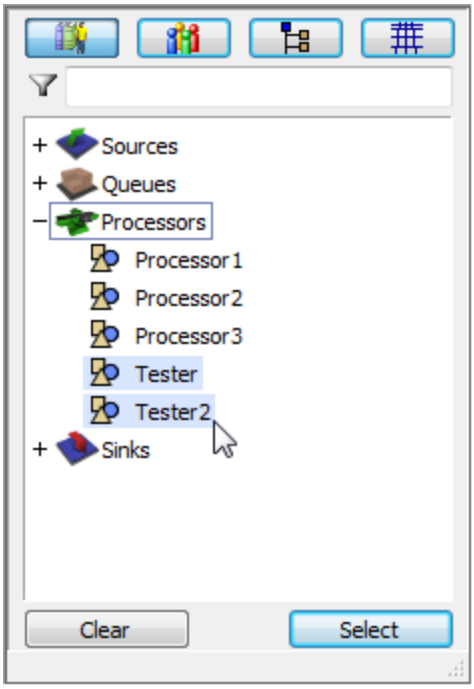
- From the **Statistics** menu at the top, select **Dashboards > Add**. The **Dashboard** window will appear.



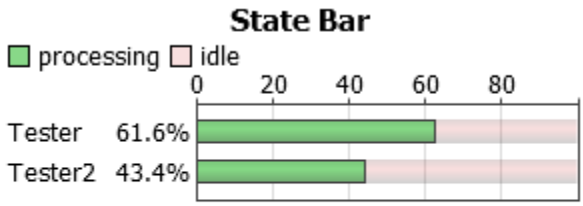
- Drag the icon into the Dashboard window. This should bring up an object selection window.



- On the **Objects** tab, click the . This will open a popup.
- In the popup, click the , expand **Processors**, and select *Tester* and *Tester2*.



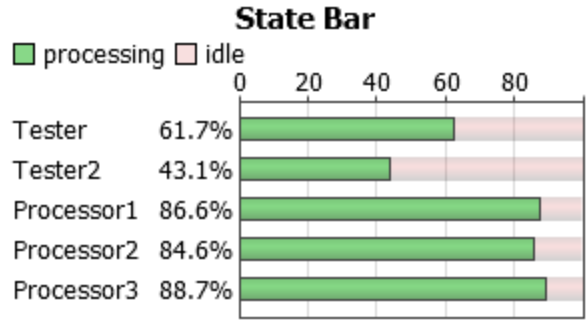
- Click the **Select** button on the popup to finalize your selection. Then click **OK**. A blank chart should appear in the dashboard.
- Reset and run the model again. The graph in the Dashboard will dynamically update.



The reason that these two are different is because the tester queue sends to the first available tester. Whenever both testers are available, a product will always go to the original tester, since it is the first available. Products only go to the second tester if the original tester is already busy. Thus the original tester gets higher utilization than the second tester.

Now add the other three processors to the **State Bar** graph.

- Double-click on the graph in the dashboard and the same object selection dialog opens.
- Select *Processor1*, *Processor2*, and *Processor3* from the selection list. Whatever you select is added to the previous contents of the graph.
- Reset and run the model again. Now all five processors can be compared side by side.



We have effectively moved the system bottleneck from the tester to the three upstream processors. Also, by increasing throughput by 15% and consequently adding another tester, we have significantly decreased the utilization of each tester. Whether this is a good decision depends much on the cost it would take to add a second tester. Since the bottleneck is in the 3 processors, in order to further increase throughput, and thus increase the utilization of each tester, we would need to add more processors. Again, there is a cost/benefit analysis to this decision.

Try changing any parameter (like a processor's process time) and watch its effect on the model. Even small changes can dramatically change the overall model.

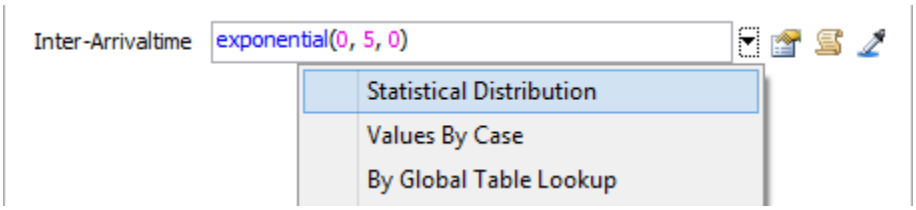
Randomness

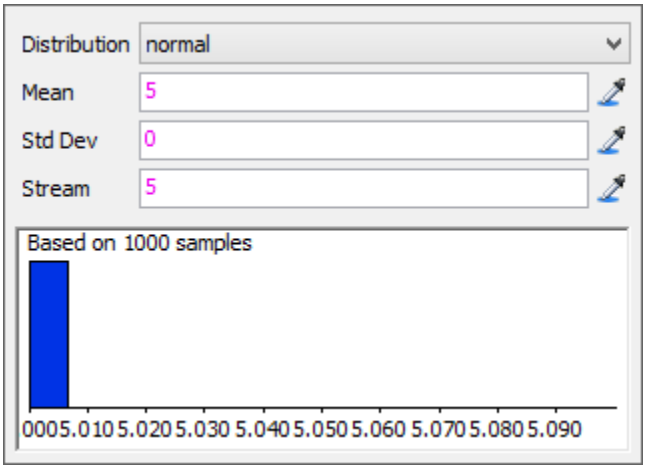
Let's do some more testing before we actually decide to add another tester. Since on average one product arrives from the source every 5 seconds, and on average one product goes to the sink every 5 seconds, why should the queue accumulate at all? Products are leaving just as fast as they arrive, so there shouldn't be any accumulation in the system.

The reason the queue accumulates is because of randomness in the system. Yes, on average a product arrives every 5 seconds, but this arrival rate is according to an exponential distribution. For an exponential distribution with a mean of 5, most of the time products will actually arrive at a faster rate than every 5 seconds. But every once in a while there will be a long drought where no products arrive at all. In the end it evens out to an average of 5 seconds, but usually products arrive faster, and thus will accumulate in the tester's queue, since the tester is the bottleneck.

What if, in our facility, products actually arrive at a more predictable rate, instead of by the somewhat unpredictable exponential distribution? Will the queue size generally stay at a lower level? Let's test it.

- Edit the *Source's* **Inter-Arrivaltime** to match the following.





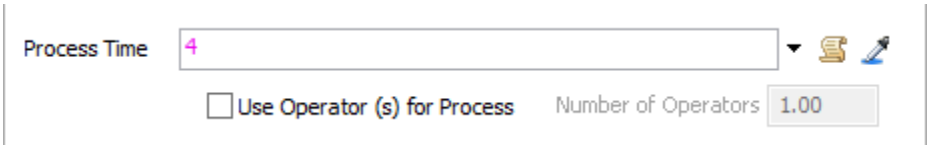
- Once set, **Reset** and **Run** the model again.

If you do not still have *Queue2*'s properties window available, open it again by double-clicking on *Queue2*. Continue to run the model. You will notice here that the queue's maximum content doesn't go up as high. Usually they won't go much higher than 50 or 60 now, whereas before they would sometimes get up to 150 or 200. This is a significant difference caused by simply changing the type of randomness in the model.

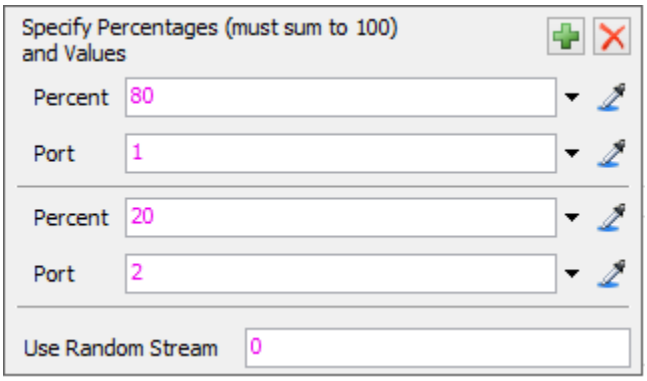
Higher Throughput

Now suppose that the facility does indeed need to increase the throughput rate of this system by 15%. This equates to a change of the mean inter-arrival time of the source from 5 seconds to 4.25 seconds. Since the tester was already at 100% utilization, we will obviously need to add a second tester to the system. Let's make this change.

- Edit the *Source's* **Inter-Arrivaltime** to be a normal distribution with a mean of 4.25.
- Now we will create a second tester. Create another **Processor** object in the model, and place it below *Tester*. Name it *Tester2.*
- Connect *Queue2* to *Tester2*.
- Connect *Tester2* to *Sink* and to *Queue1*.
- Set *Tester2's* **Process Time** to 4.



- Change *Tester2's* **Sent To Port** to **By Percentage** Enter the same parameters as you did for *Tester1*.



- Add an OnExit trigger to change the color to black, just like the other Tester.
- Now that you have finished making the changes, **Reset** and **Run** the model again.

Results

By creating a model that simulates our system, we have clearly determined what effect certain decisions will have on the system. We can now use the information we have gathered from the simulation to make better informed decisions for the future of the facility.

With this simple model, many of the same conclusions could have been made through mathematical models and formulas. However, real systems are often much more complex than the model we have just built, and are outside the scope of mathematical modelling. By using FlexSim simulation, we can model these real-life complexities, and examine the results just as we have done in this model.

FlexSim also gives your simulations much more visual appeal. It is much easier to convince a management team of the wisdom in a decision if the management team can see the effects of that decision in a virtual 3D world. This world is created automatically as you build your FlexSim models.

What Next?

Now that you have become familiar with FlexSim and the use of simulation, we suggest that you go through the other tutorials included in FlexSim Help.

License Activation

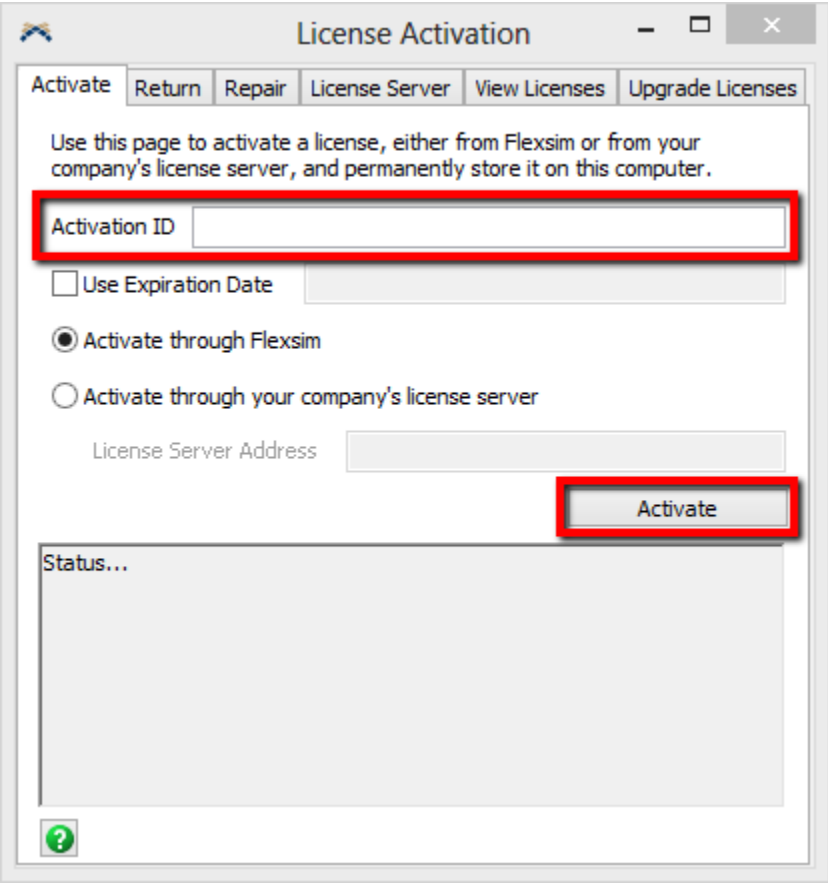
1. Concepts
2. Example
3. Reference

License Activation Concepts

The License Activation window is found in the main menu option **Help > License Activation**. From this window, you can activate standalone licenses, return licenses, repair licenses, configure the client to use a concurrent license server, view license rights held in Flexnet trusted storage, and upgrade standalone licenses.

Topics

- **Licensing**
- **License Server**
- **Repairing Licenses**
- **View Licenses**
- **Upgrading Licenses**



Licensing

FlexSim does not require a license for it's trial version. The trial allows you to create 20 objects in your model and run that model with various scenarios.

If you would like to purchase a license for this software you may contact our sales department at (801) 224-6914 or email us at sales@flexsim.com

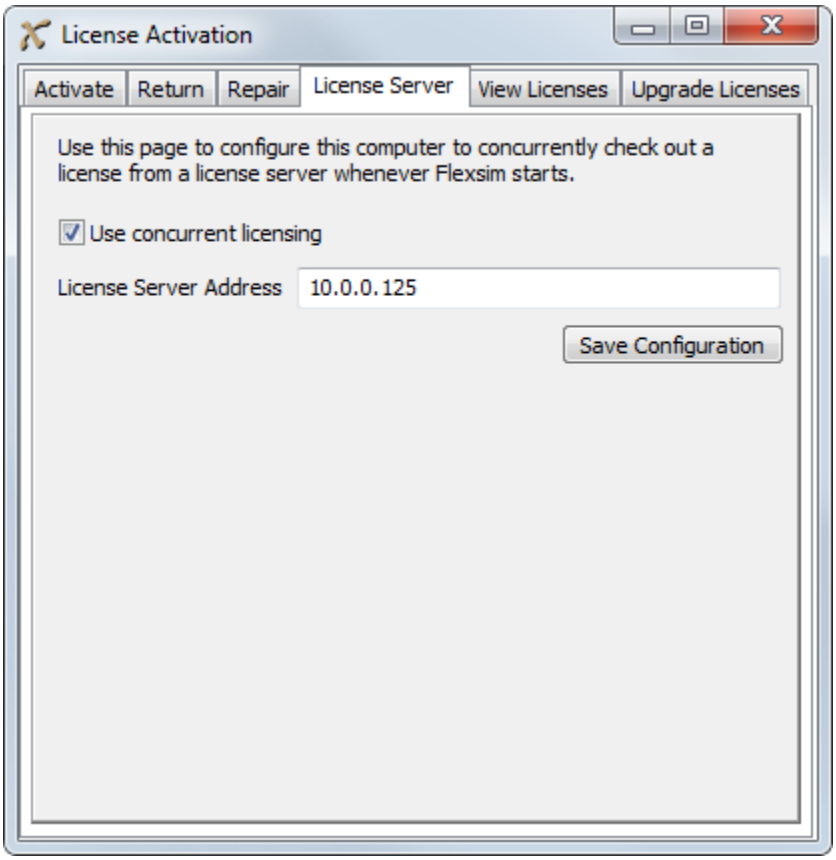
Once you have successfully activated a license from either your company's server or the FlexSim license server, many of the menu options that were grayed out in the demo mode will be available for use. You will also see your license type in the **Help > About FlexSim** window or the Start Page.

For an explanation of errors received while activating/returning licenses, visit the reference page.

Repairing Licenses

On the repair tab, you will see any disabled licenses on your computer. Licenses are held in Flexnet trusted storage on a computer. Certain information about the licenses are stored in various places on your hard disk and registry. Tampering with these locations may break the license trust flags, which disables the license. This may happen with some registry cleaners or Windows restore points. If your license was disabled, then you can use the repair tab to generate an xml repair request file. Email that file to FlexSim support using the website (www.flexsim.com/support) and then you will be emailed back a repair response xml file to process on the repair tab. Processing the repair response from FlexSim's support team will repair the trust flags on your disabled license and allow it to be used again.

License Server



If you are using a concurrent server license, you can configure the clients to connect to the server using this tab. Check the box and enter the license server ip address. Press the **Save Configuration** button, then close and reopen FlexSim in order to check out a license from the server.

Before attempting to connect the clients to a concurrent server, you must activate your licenses on the server and start a License Server Manager program on the server. Instructions and files necessary for configuring the server are available in the LAN License Tools download available in the account section of the FlexSim website if you have a concurrent server license on your FlexSim account.

The Windows registry entry for FLEXlm, our license manager, stores any locations that FlexSim has looked for a license server or file, and it can store multiple locations. So, whenever you enter a new value in the Concurrent License Server Address and restart FlexSim, it adds more entries to the registry. If you have successfully checked out a license from a server, FlexSim may still successfully check out a license from that server when you start FlexSim, even if the "Use concurrent licensing" box is unchecked.

For help with troubleshooting concurrent server errors, visit the reference page.

View Licenses

On the view licenses tab, you can see the contents of this computer's Flexnet trusted storage.

Licenses are not in any way tied to any FlexSim installation. Your computer itself is licensed. The actual FlexSim install or version doesn't matter at all. One license can license every FlexSim program on the computer. You simply use the FlexSim program to activate/return/view licenses on the computer for simplicity. The licenses are not tied to an installation of FlexSim.

For example, you could install FlexSim, activate a license, uninstall FlexSim, and then install FlexSim again, and your license will still be there. The FlexSim programs are entirely separate from where licenses are stored on your computer in Flexnet trusted storage.

It doesn't matter what version of FlexSim you use to activate or return licenses; they are calling exactly the same code.

You can have just one license of 5.1 on the computer and that license will work with both 5.0.4 and 5.1.0 just fine. You don't need multiple licenses to license multiple FlexSim programs. The license is not tied to the installation. The license is tied to the computer. Any FlexSim programs on a licensed computer will work. However, a computer that has a 5.0 license won't be able to run 5.1 software. The computer's license must be >= the version number of the software in order to run.

If you uninstall FlexSim without first returning your license, you will need to reinstall Flexnet in order to return the license so that you can move it to another computer. Because licenses are tied to the computer, not the FlexSim installation, you do not need to return your license if you are simply uninstalling and reinstalling FlexSim.

Upgrade Licenses

On the upgrade licenses tab, you can request upgrades to your licenses. This is necessary to run newer versions of the software. For instance, if you previously had FlexSim 5.0 installed and you upgraded to 5.1, you would need to also upgrade your license to 5.1.

The process to upgrade a license to a newer version has three steps:

1. The FlexSim License Server needs to be told to upgrade the license, which creates a new Activation ID for the upgrade.
2. The old license needs returned to the FlexSim License Server.
3. The new license needs activated on the client machine.

This process can be done manually by returning any old licenses, pressing the Upgrade Licenses button on the "My Licenses" page in the account section of the FlexSim website, and then activating the new Activation IDs on that page (you may need to refresh the licenses page table).

The **Request Upgrades** button on the Upgrade Licenses tab automates this process into a single button click by sending http requests to the FlexSim server. If this automatic process doesn't work, then you can do it manually instead.

To upgrade server licenses, you must manually return the licenses and then activate the new licenses.

License Activation Example

Topics

- **Activating a Standard License**
- **Activating through your License Server**
- **Returning a License**
- **Repairing a License**
- **Activate a Concurrent License**

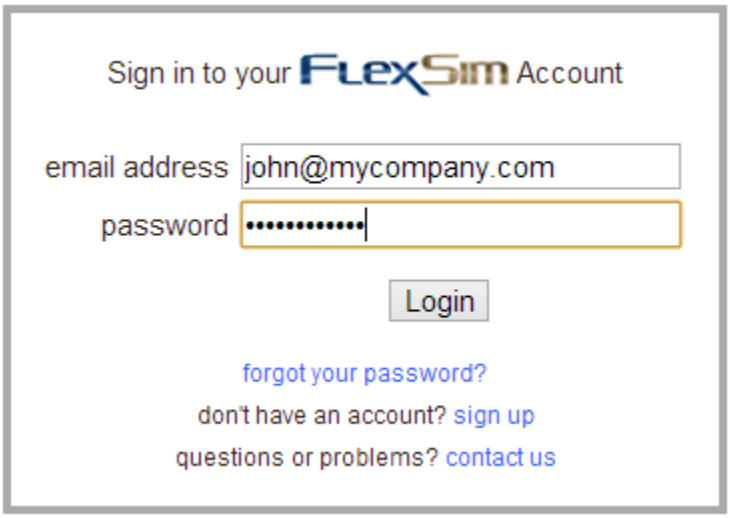
Activating a Standard License

This will take you through the steps necessary to activate a standard FlexSim license through the FlexSim License Server. If your company has it's own license server, see the Activating through your License Server section. If your company is using a concurrent license server, see the Activate a Concurrent License section.

Login to your FlexSim.com account

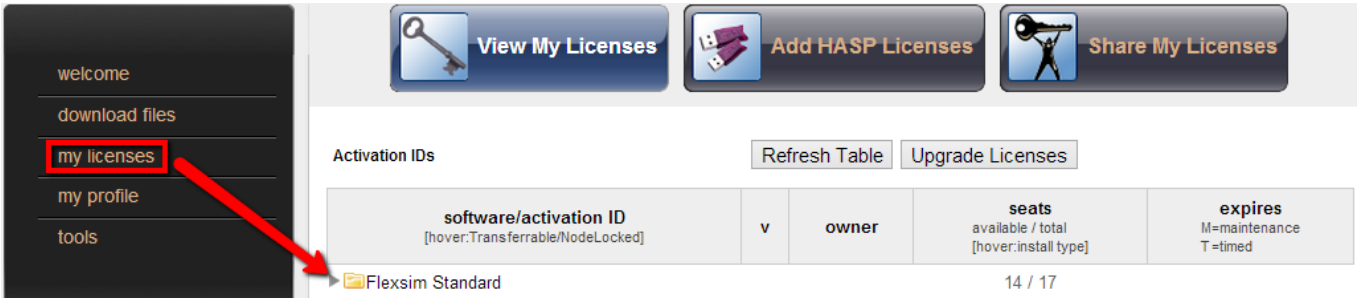
First you'll need to retrieve an available license from your FlexSim account.

- Go to http://www.flexsim.com/account/ and login.



Find an available license

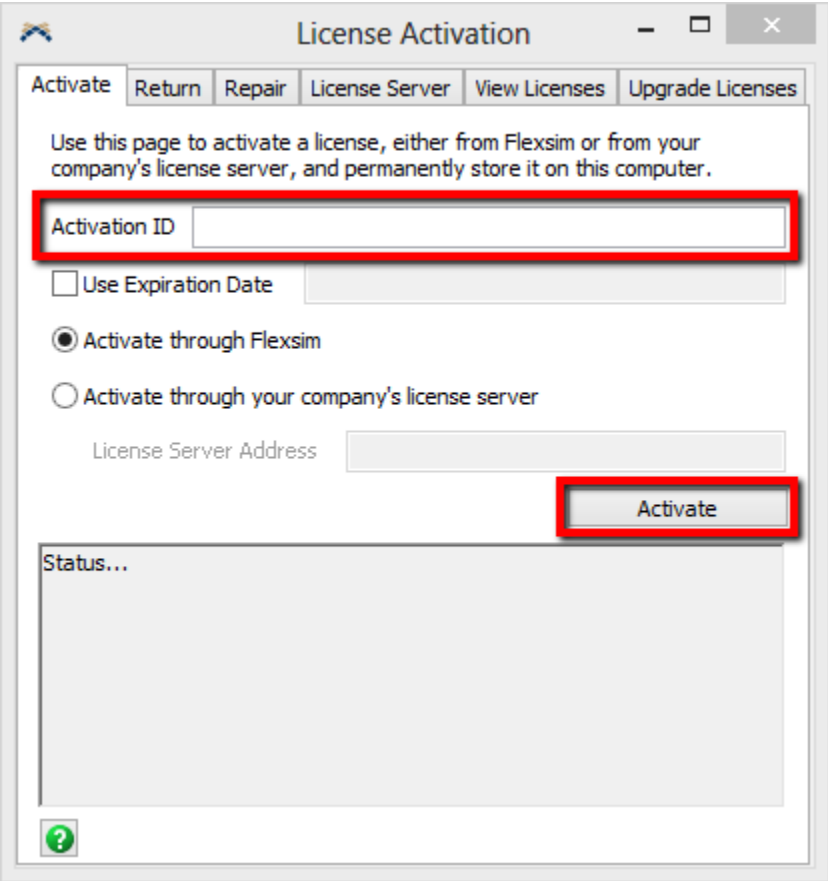
- Once logged in, go to the 'my licenses' page.
- Expand your Flexsim Standard folder to show your available licenses.



You can see the number of seats available to the right, as well as expiration dates for timed and maintenance licenses.

Activate the license

- Copy the Activation ID for an available license. The activation ID will start with flexsim.com.
- Paste the Activation ID into the Activation ID field.
- Press the **Activate** button.

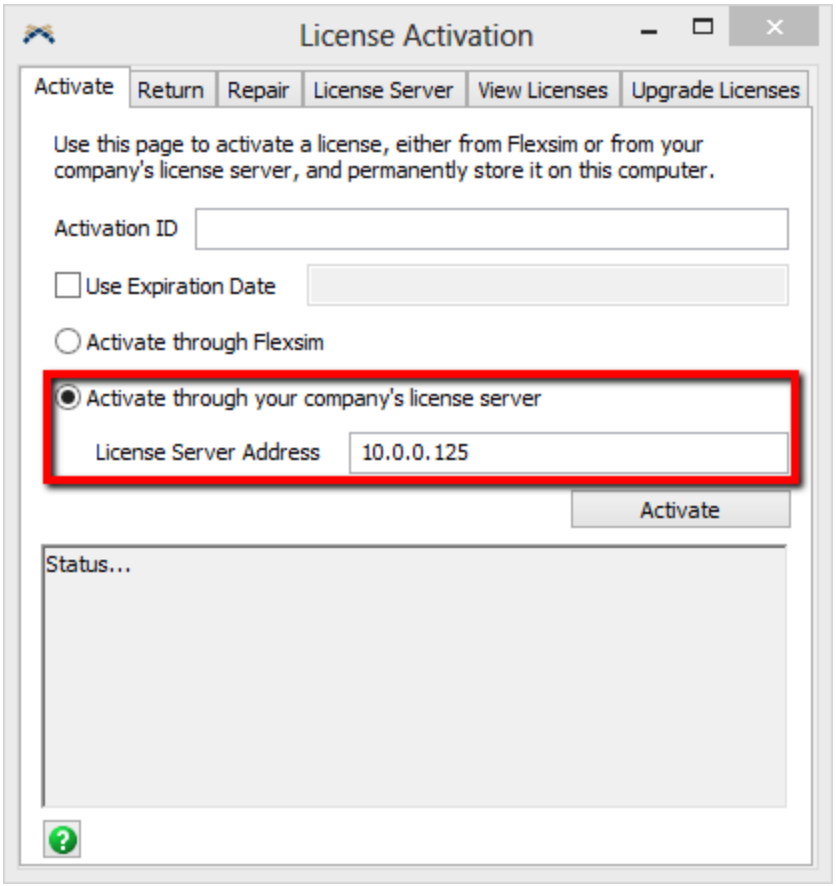


Note: If your license is timed, then you may need to check the Use Expiration Date box and enter the expiration date in the format 23-NOV-2009 before activating.

For an explanation of errors received while activating licenses, visit the reference page.

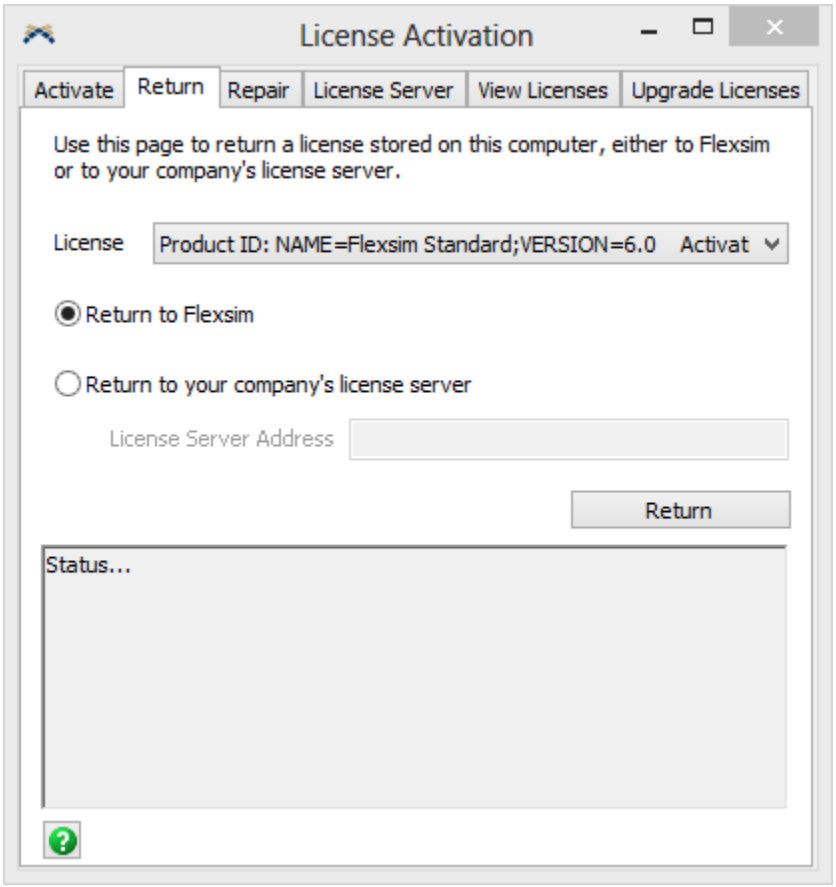
Activating through your License Server

- In the Activate tab of the License Activation window, check the **Activate through your company's license server** button.
- Enter the ip address of your company's license server in the **License Server Address** field.
- Press the **Activate** button.



For an explanation of errors received while activating licenses, visit the reference page.

Returning a License



Returning a Standard License

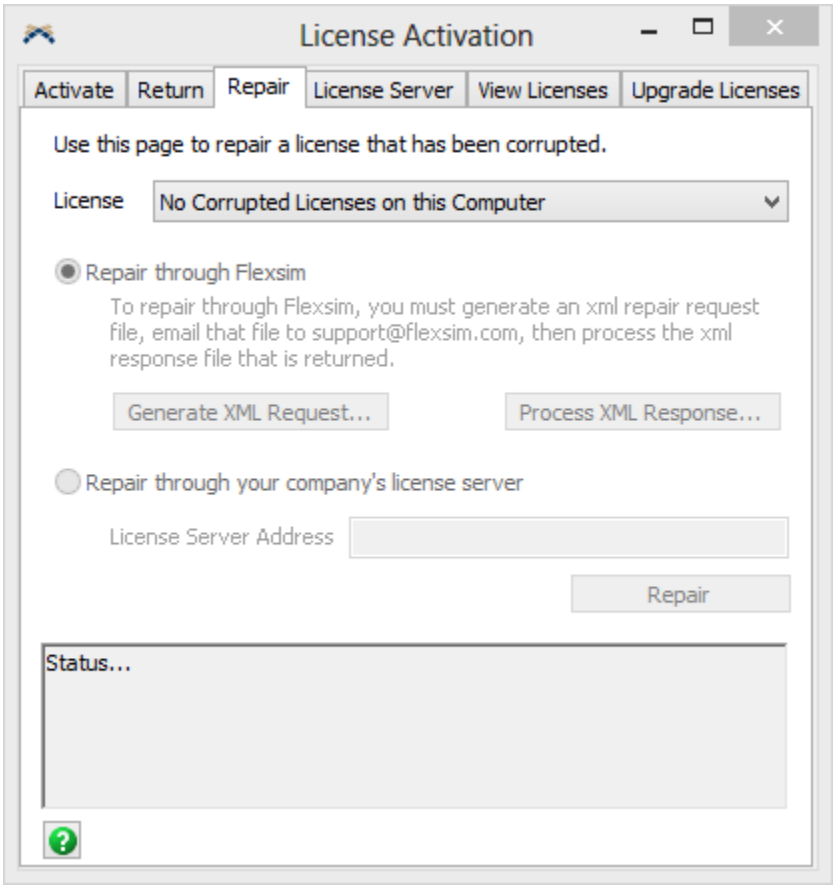
- Select the license you want to return from the **License** dropdown menu.
- Press the **Return** button.

Returning to your company's License Server

- Check the **Return to your company's license server** button.
- Enter the ip address of your company's license server in the **License Server Address** field.
- Press the **Return** button.

For an explanation of errors received while returning licenses, visit the reference page.

Repairing a License


License Activation
Activate
Return
Repair
License Server
View Licenses
Upgrade Licenses
Use this page to repair a license that has been corrupted.
License
No Corrupted Licenses on this Computer
Repair through Flexsim
To repair through Flexsim, you must generate an xml repair request file, email that file to support@flexsim.com, then process the xml response file that is returned.
Generate XML Request...
Process XML Response...
Repair through your company's license server
License Server Address
Repair
Status...

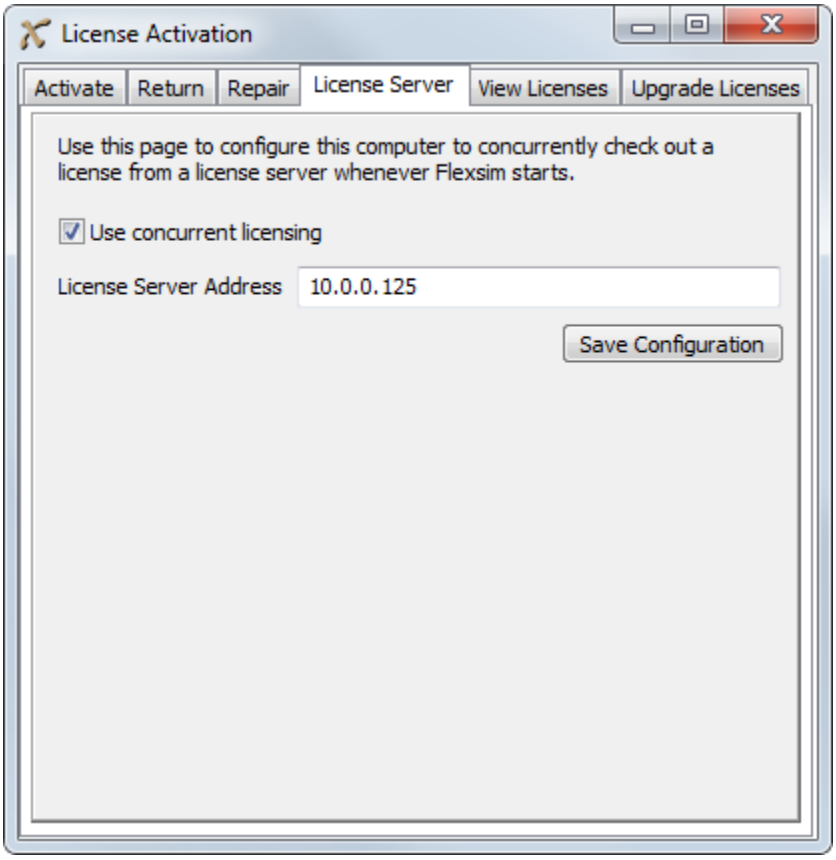
Repairing a Standard License

- Select the corrupted license from the **License** dropdown menu.
- Press the **Generate XML Request...** button.
- Email the request file to support@flexsim.com
- Once the FlexSim support team sends you the XML response file, press the **Process XML Response...** button and select the response file.
- Press the **Repair** button.

Repairing a license through your company's License Server

- Check the **Repair through your company's license server** button.
- Enter the ip address of your company's license server in the **License Server Address** field.
- Press the **Repair** button.

Activate a Concurrent License



- Check the **Use concurrent licensing** button.
- Enter the ip address of your company's license server in the **License Server Address** field.
- Press the **Save Configuration** button.

For more information on setting up a concurrent license server, see the documentation in the LAN License Tools.

License Activation Reference

Topics

- **Common Activation Errors**
- **Common Return Errors**
- **Troubleshooting Concurrent License Server Errors**

Common Activation Errors

Operations Error 7288

```
Status...
Status: 4 Creating request
Status: 5 Request created
Status: 6 Context created
Status: 7 Connected to remote server
Status: 8 Request Sent
Status: 9 Polling for response
Status: 10 Waiting for response
Status: 9 Polling for response
Status: 11 Done
Operations error: 7288  The activation of the fulfillment is denied by the activation policy because fulfill count exceeded the available seat count.
```

Operations error 7288 means that your license has already been successfully activated on a computer. In order to activate the license, you must find the computer that contains the license and return it. You can find the computer that contains the license by checking the output of View Licenses on any computers that you may have activated the license on.

Error 50041

```
Status...
Status: 4 Creating request
Status: 5 Request created
Status: 6 Context created
Status: 0 Error
ERROR: flxActAppActivationSend - (50041,41143,34)
"Failed to connect to the license server or Operations server.
Recovery: check connection and that server is operational."
```

Error 50041 usually means that your computer is not connected to the internet or your network/firewall settings are preventing communication with the FlexSim license server. The license activation mechanism uses soap requests sent through http port 80. Make sure this type of communication is available on your network.

Common Return Errors

Operations Error 7466

```
Status...
Status: 4 Creating request
Status: 5 Request created
Status: 6 Context created
Status: 7 Connected to remote server
Status: 8 Request Sent
Status: 9 Polling for response
Status: 10 Waiting for response
Status: 9 Polling for response
Status: 11 Done
Operations error: 7466  The return of the fulfillment is denied by the return policy because max return exceeded
```

Operations error 7466 means that returns are disabled on your license. Your license is configured to be a one-time activation onto a computer. If you believe your license should be able to transfer from one computer to another, contact your local distributor or FlexSim support to discuss the situation.

Troubleshooting Concurrent Server Errors

1. Make sure the client computer is connected to the server on a local area network. Be sure you can ping the server at the address specified.
2. Make sure that the server has valid licenses using the menu option **Tools > View License Rights** in the flexsimserveractutil.exe program on the server. This program is contained in the LAN License Tools available for download on the FlexSim website.
3. Make sure that the server's License Server Manager program (lmadmin or lmtools) is properly configured and has licenses available for use.
4. Consult the Flexnet License Administration Guide for additional details on how Flexnet server licenses work. This manual can be found at http://www.globes.com/support/fnp_utilities_download.htm.
5. If you still are still having troubles, contact FlexSim Support. Be sure to send them the information that is printed when you press Tools > View License Rights in the flexsimserveractutil.exe program and also a screenshot of the Dashboard view in lmadmin (or the Status Enquiry output from lmtools).

FlexSim Concepts

1. Overview
2. Flowitems
3. Ports
4. Order of Events
5. Itemtype
6. Labels
7. Item and Current
8. Return Values
9. Picklists
10. Template Code
11. Model Tree View

FlexSim Concepts Overview

This topic describes, in detail, concepts that are essential to understanding how to build models in FlexSim. You should have gone through the getting started and tutorial models provided within this user manual before reading this topic. In this topic we will not build any specific models. This allows us to focus exclusively on the concepts being discussed, instead of spending time on model building steps. We will, however, cite an example model where the concepts can be applied, and please feel free to build your own model as you go along in this topic. If you have gone through the tutorials, you should have the skills needed to build the model examples cited. If you do decide to build your model, however, I would advise that you read through this whole topic once, then go back and build the model as you go, because there are some things at the end of the topic that you'll want to understand before building the model.

Introducing... FlexScript

Don't let the name fool you – **FlexScript** is a powerful yet easy-to-learn scripting language that will help breathe life into even the most complex simulation model. Unique to FlexSim's line of software solutions, FlexScript provides a straightforward approach to allow users to quickly customize triggers and parameters within simulation projects.

Throughout the Concepts sections we will often include snippets of code that help clarify the concept being discussed. The logic that the example code implements can be done in other ways by using the drop-down pick options, but we want to help you become more familiar with FlexScript and will therefore use straight FlexScript code examples. If you are still unfamiliar with FlexScript, then you can skip those example snippets and move on, but we try to give a concise description of what the code samples do, so you will hopefully be able to understand what's going on even if you are new to FlexScript.

For more information on writing FlexScript code refer to the topic on writing logic in FlexSim.

Flowitems

Flowitems are the simple objects that are created to move through the model. They can represent actual objects, or they can be representative of a more abstract concept.

Flowitems are copied into the model from the Flowitem Bin. You can learn more about the Flowitem Bin in the Modeling Tools - Flowitem Bin section.

Flowitems only remain in until the model is reset, then all flowitems are destroyed.

Ports

Every FlexSim object has an unlimited number of ports through which they communicate with other objects. There are three types of ports: input, output, and central.

Input and output ports are used in the routing of flowitems. For example, a mail sorter places packages on one of several conveyors depending on the destination of the package. To simulate this in FlexSim, you would connect the output ports of a Processor object to the input ports of several Conveyor objects, meaning once the Processor (or mail sorter) has finished processing the flowitem (or package), it sends it to a specific conveyor through one of its output ports.

Central ports are used to create references from one object to another. A common use for central ports is for referencing mobile objects such as operators, fork lifts, and cranes from fixed resources such as machines, queues, or conveyors.

Creating ports

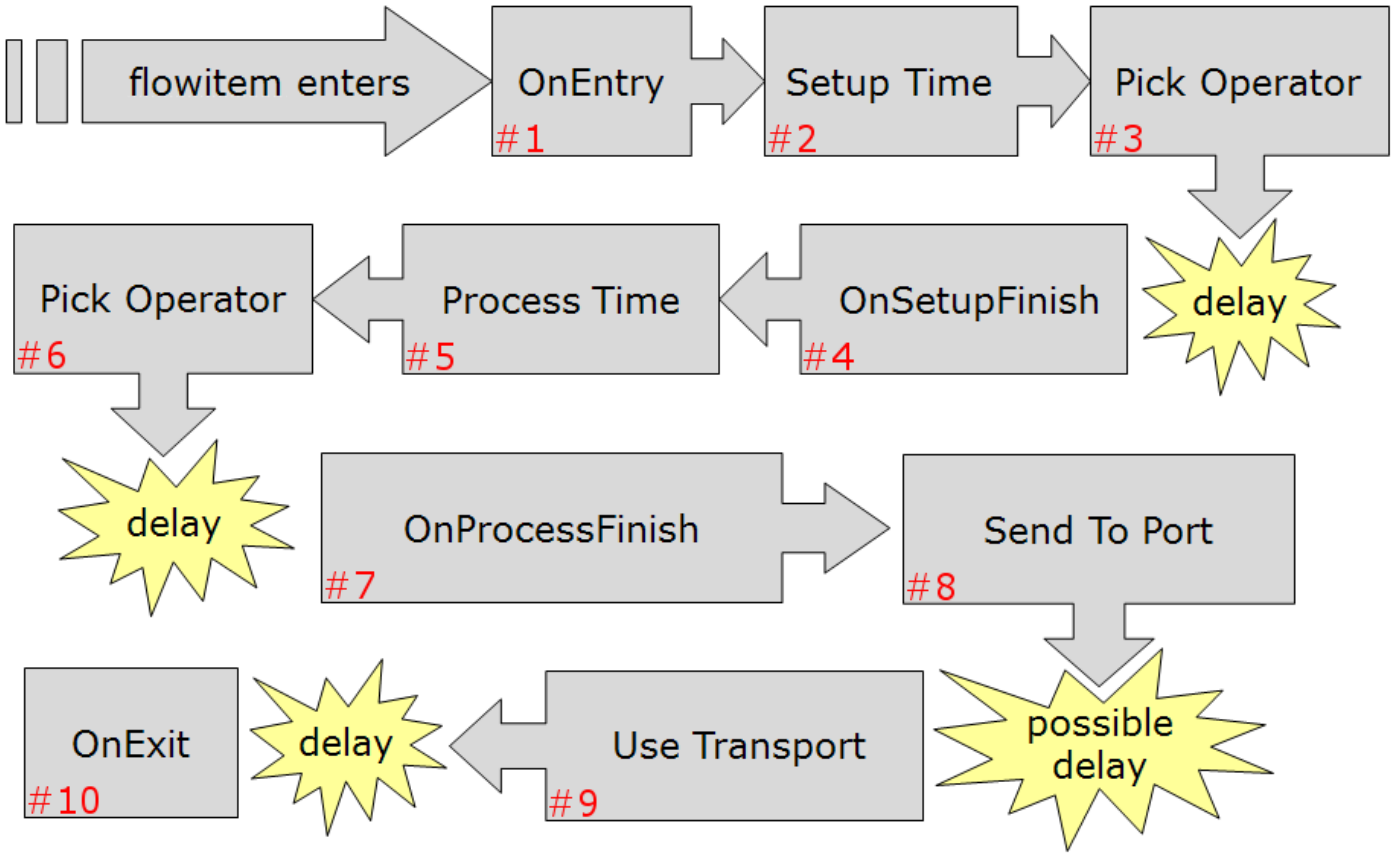
Ports are created and connected in one of two ways:

1) By clicking on one object and dragging to a second object while holding down different letters on the keyboard. If the letter 'A' is held down while clicking-and-dragging, an output port will be created on the first object and an input port will be created on the second object. These two new ports will then be automatically connected. Holding down the 'S' key will create a central port on both objects and connect the two new ports. Connections are broken and ports deleted by holding down the 'Q' for input and output ports and the 'W' key for central ports. The following table shows the keyboard letters used to make and break the two types of port connections. The First Model from the Getting Started section demonstrates how to properly make port connections.

	Output - Input	Center
Connect	A	S
Disconnect	Q	W

2) By entering the connection mode via clicking the button. Once in the connection mode, there are a couple of ways to make a connection between two objects. You can either click on one object, then click on another object; you may also click and drag from one object to the next as with method one. Either way, keep in mind that the flow direction of a connection is dependent on the order in which you make the connection. Flow goes from the first object to the second object. Incidentally, connections can be broken by clicking the button then clicking or dragging from one object to another in the same manner as when you connected them. Center port connections are not affected by the order in which the objects are connected.

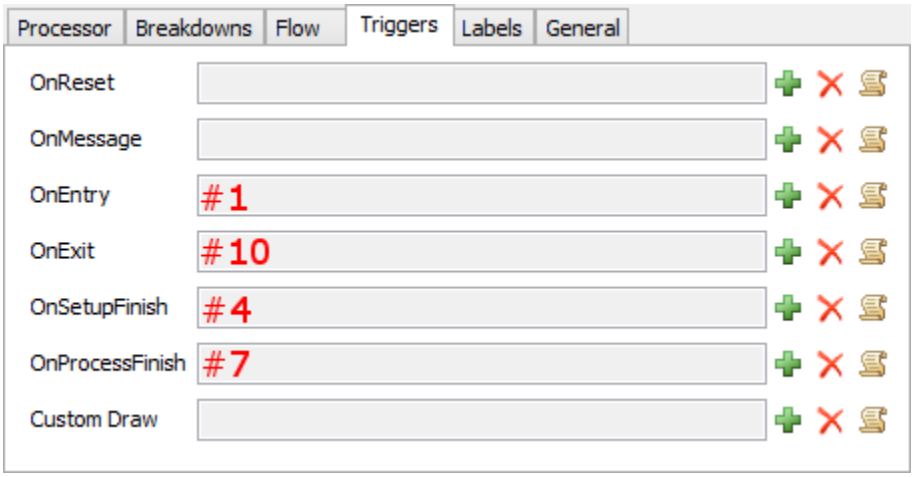
Order of Events

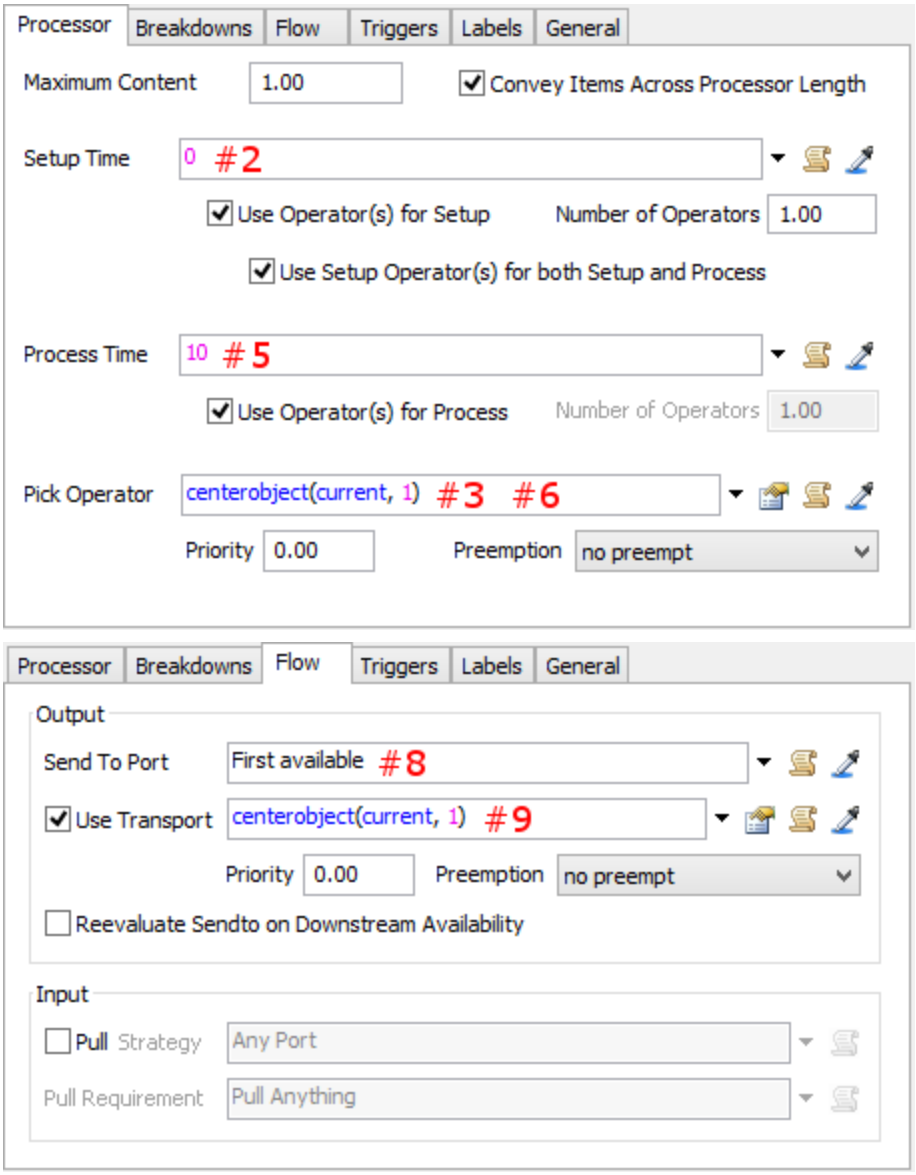


As flowitems move through your model, they enter and exit Fixed Resource objects. Each flowitem will follow a path similar to the one displayed above. This path is for a Processor object.

Example

Consider a flowitem entering a Processor. The following images show where each trigger and picklist can be accessed through the Processor's Properties window. Some of these can also be accessed through the Quick Properties.





Step 1: OnEntry

When the Entry Trigger fires, the flowitem has already been moved into the Processor object, so the content of the Processor increases by one. Code can be added to the OnEntry trigger to customize what occurs when the flowitem enters.

Step 2: Setup Time

The next segment of code to fire is the Setup Time. This picklist gives you the option to define the number of model time units it takes for the flowitem to be "setup" in the Processor.

Step 3: Pick Operator

If the "Use Operator(s) for Setup box is checked, the Pick Operator code will fire, calling to a TaskExecuter object. There is a delay associated with calling an operator as the operator must move from their current location to the Processor. The delay may lengthen if there are no available operators when the Pick Operator code is fired.

Step 4: OnSetupFinish

Once setup is complete, the Setup Finish Trigger will fire.

Step 5: Process Time

Similar to the Setup Time, this picklist allows you to define a Process Time for the flowitem.

Step 6: Pick Operator

If the "Use Operator(s) for Process" is checked, the Pick Operator will fire. If operators were used for the Setup and Process, and "Use Setup Operator(s) for both Setup and Process" is unchecked, then this picklist will fire twice, once to call a setup operator and then again to call a process operator. Otherwise, this picklist will only be called once.

Step 7: OnProcessFinish

Once setup is complete, the Process Finish Trigger will fire.

Step 8: Send to Port

At this point, the flowitem is ready to leave the Processor, or FixedResource object. The Send To Port will fire and attempt to find a destination for the flowitem. The possible delay happens if the object is unable to find an available destination for the flowitem. This typically happens when downstream objects are full and have no available space to accept the flowitem. It may also be because downstream FixedResource objects are stopped, or their inputs are closed.

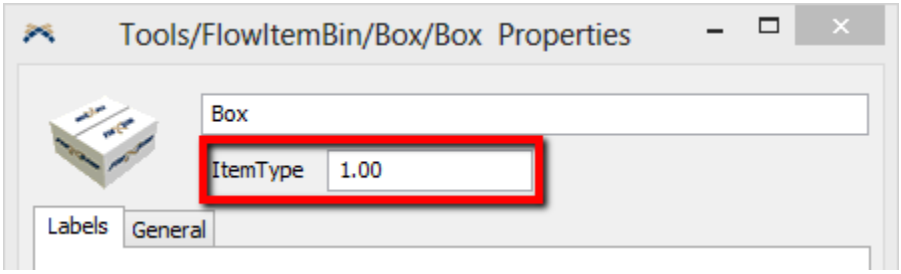
Step 9: Use Transport

If "Use Transport" is checked, and once a destination has been founding using the Send to Port, the Use Transport picklist with fire to call a Task Executer to the Processor to transport the item from the Processor to its destination. Again, this will result in a delay as an available operator is found and moves to the Processor.

Step 10: OnExit

Finally, the Exit Trigger fires. When this trigger fires, the flowitem has not yet exited the Processor, so the content of the processor remains unchanged. Once this trigger is complete, the flowitem will be moved from the processor (decreasing its content by 1) and into the Task Executer or downstream Fixed Resource object. If you need code to be executed when an item leaves a FixedResource, but it needs to be fired immediately following the item actually be moved from the object, call senddelayedmessage() with a delay time of 0.

Itemtype



Itemtype is a value that is stored on every flowitem that travels through a FlexSim model. This value can be accessed and/or changed at any point in the flowitem's progress through the model. Every single flowitem has its own unique itemtype value, meaning that if you change the itemtype of one flowitem, it will only change the value for that specific flowitem, and the itemtype of other flowitems will not be changed. The meaning of the itemtype value is completely up to you the modeller. In general it is meant to be a value that describes a product type or category.

Example

Take for example a post office where customers come in to either have a package sent or to have copies made. In this model the flowitems, or customers, are separated into two general categories, namely those who need a package sent and those who need copies made. The itemtype value can be used to make this designation. For example, customers, or flowitems, who need a package sent can be given the itemtype value 1, and customers who need copies made can be given the itemtype value 2. In FlexSim, you will usually set the itemtype value in the Source object when the flowitem is first created. In our example let us say that 60% of arriving customers are "package" customers, and 40% are "copy" customers. To do this in FlexSim we would go to the Source's OnExit trigger and access the Code Editor by clicking the . Then add the command:

```
setitemtype(item, bernoulli(60, 1, 2));
```

Source1 - OnExit*

```
1 treenode item = parnode(1);
2 treenode current = ownerobject(c);
3 int port = parval(2);
4
5 setitemtype(item, bernoulli(60, 1, 2));
```

Alternatively, there is a picklist option available for setting the itemtype by pressing the next to the OnExit trigger and selecting *Set Type, Name or Label, or Set Item Type and Color.*

This command will randomly set the itemtype value of the flowitem that is exiting to 1 60% of the time and to 2 40% of the time. The **setitemtype** command sets the itemtype value of an object. It takes two parameters. The first parameter is a reference to the flowitem that you want to set the itemtype value on, and the second parameter is the value to set it to. In this example, the first parameter is "item", or the flowitem that is currently exiting the Source, since we are in the Source's exit trigger (the reference "item" will be discussed in more detail later).

Notice that for setitemtype's second parameter, the **bernoulli** command is used. This command takes 3 parameters and returns one of two possible values. The first parameter is a percentage value between 0 and 100. The second and third parameters are "success" and "failure" values, and represent the two possible values to be returned. In this case, 60% of the time the bernoulli command will return a 1 (parameter 2) and the rest of the time it will return a 2 (parameter 3). Since the bernoulli command is the second parameter of the setitemtype command, the itemtype value will be set to whatever is returned by the bernoulli command, namely the 1 or 2 value. The exit trigger of the Source is executed every time a flowitem is created and exits the Source. This means that the setitemtype command will be executed many

times over the course of the simulation, and each execution will be associated with exactly one flowitem. Items exiting the Source will thus be split 60/40 for itemtypes 1 and 2 respectively. Since bernoulli is a stochastic, or random, command, the bernoulli will not always keep a perfect 60/40 ratio. You may have several consecutive customer arrivals whose itemtype value is set to 1, and vice versa. But over time the ratio will equalize out to 60/40.

Once you have initialized the itemtype value, logic in your model can then be implemented based on the itemtype value of each customer going through the model. In the example, a "package" customer may take 3 minutes to service, whereas a "copy" customer may take 5 minutes to service. In FlexSim, you would implement this difference by writing FlexScript code in the **Process Time** picklist of a **Processor** object. The code would look like this:

```
if(getitemtype(item)==1) return 3;


else return 5;
```

Processor2 - Process Time*

```
1 treenode current = ownerobject(c);
2 treenode item = parnode(1);
3
4 if (getitemtype(item) == 1) return 3;
5 else return 5;
```

This code basically says: if the itemtype value of the flowitem that is currently being serviced (getitemtype(item)) is equal to 1 (it is a "package" customer), then return 3 as the process time. Otherwise (it is a "copy" customer) return 5 as the process time.

Again, this example can be done without writing any code, using FlexSim's pick-lists to define your logic. Nevertheless, the key concept to understand here is that every flowitem can have an itemtype value whose meaning is up to you, and that you can use the itemtype value to make decisions in your model.

Note on FlexSim objects: Every flowitem in a model has an itemtype. However, FlexSim objects like Sources, Queues and Processors do not have an itemtype.

Note on the itemtype value: The itemtype is a double precision floating point number. This means that the itemtype can not only hold integer values like, 1,2,3, etc., it can also hold floating point values like 1.5 or 99.9. However, the itemtype cannot hold string values like "package".

Note on flowitem appearance: The itemtype value will not define the visual appearance of the flowitem. This can be set by choosing the flowitem class in the Source's Parameters window, such as box, tote, or pallet.

Labels

Labels are also a key concept to understand in building models in FlexSim. They are very similar to the itemtype value in that they store data on objects that can be used in making decisions in the model. However, there are some key differences, listed below:

- Each label has a name that is defined by you the modeler.
- Unlike itemtype, which is specific to flowitems, labels can be defined on objects as well as flowitems (e.g. Sources, Queues or Processors).
- An object can have as many labels as you choose to give it.
- Labels can have number or string values, whereas the itemtype can only have a number value. Labels can even hold lists or tables of values.
- With labels, you must explicitly add the label to the object through its properties window, unlike the itemtype value, which is automatically included with every flowitem.
- When adding a label to a flowitem in the Flowitem Bin, the label is specific to that flowitem class. This means that if you add a label to the Pallet flowitem class, only flowitems that are created from that Pallet class will have that label on them.

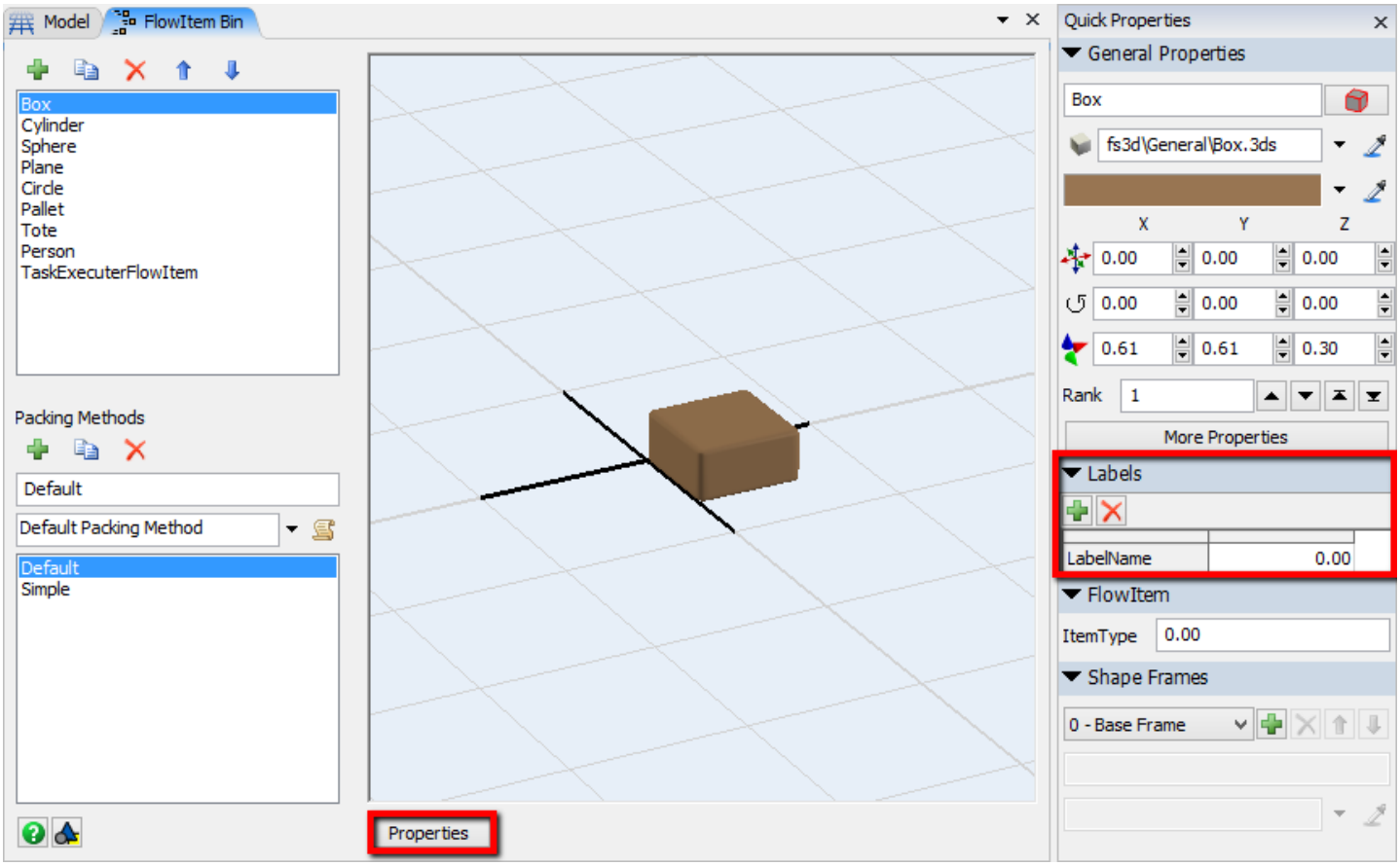
Values

For flowitems, the value you specify for labels will be the default value for all flowitems that are created, but you can change that value on each flowitem as it progresses through your model. For FlexSim objects' labels, the label's value will remain the same unless you either have logic within the object that changes the label's value.

Label values will not reset on their own unless you check the **Automatically Reset Labels** button in the label tab page. Alternatively, you can add code or a picklist option to the object's OnReset trigger to reset the label value. Both options will set the label back to an initial value when you press the Reset button to reset a model.

Example

To add labels to flowitems, go to the FlowItem Bin, select the flowitem class that is being created by your Source (Box in this example), and edit the labels from the Quick Properties view. Alternatively, you can press the properties button (our double click the box) to display the Flowitems properties and edit the labels tab. You can add string or number labels to your objects.



The process works similar for other FlexSim objects. You can edit their labels from the Quick Properties window, or double-click on the object and edit and go to the labels tab. Specify each label's name in the row headers column on the left, and its value to the right of its name.

Let's extend the example model mentioned in the Itemtype section to use labels. Let's say for example that each "copy" customer that comes into the post office has a certain number of copies that need to be made, and that the service time for that customer is dependent on the number of copies needed. A customer that needs 1000 copies will take longer to service than a customer that needs 1 copy. As before, the itemtype value of each flowitem, or customer, reflects the category of customer, either "package" or "copy", but now for "copy" customers we need to add a label that tells us how many copies that customer needs. Again, to add a label to a flowitem, go to the **Flowitem Bin**, then select the flowitem class and click on **Properties**. Here we would add a number label ("Add Number Label") and give it a name like **"nrofcopies"**. As the default value we would leave it at 0 and set the value in the Source's exit trigger.

Once the label has been added in the Flowitem Bin, we can set the label's value when each flowitem exits the Source. In the example the copy customers will need a random number of copies between 1 and 1000. To implement this, you would modify the **Exit Trigger** of the **Source** as follows:

```
setitemtype(item, bernoulli(60,1,2));


if(getitemtype(item)==2)

        setlabelnum(item, "nrofcopies", duniform(1,1000));
```

As described previously, the setitemtype command sets the itemtype of the item to a 60/40 split between 1 and 2. Now we add an "if statement". This if statement basically says: if the itemtype of the exiting flowitem is 2 (it is a copy customer), then set the value of the flowitem's label named **"nrofcopies"** to a random number between 1 and 1000. The **setlabelnum** command sets a label value and takes 3 parameters. The first parameter is the object whose label we want to set (item, or the flowitem that is exiting). The second parameter is the name of the label ("nrofcopies"). This parameter needs to be in quotes since it is a string parameter.

The third parameter is the value to set the label to (duniform(1,1000)). The duniform command returns a value from a discrete uniform distribution. It takes 2 parameters, namely the minimum and maximum value, and returns a random number between those two values, uniformly distributed, meaning every value between the min and max is just as likely to be returned as any other value between the min and max. The "discrete" part means that the command will only return 1,2,3, etc, as opposed to the uniform() command, which may return values like 1.5 or 2.5. Since there will never be a customer that needs 1.5 copies made, we use the duniform() command.

Note that by adding the "nrofcopies" label in the Flowitem Bin, every flowitem that is created from that flowitem class will have that "nrofcopies" label on it. Even package customers will have that label, but our logic will simply not look at the label if it is a package customer.

Now that we have set up our label and set its initial value, we can define logic to make decisions based on the value of that label in the model. For a copy customer, for example, we can change the service time based on the number of copies that the customer needs. For each copy customer, the service time can be a base of 5 minutes as before, plus an additional 5 seconds for each copy that needs to be made. To make this change, you would again go to the Processor's **Process Time** field and change it to the following:

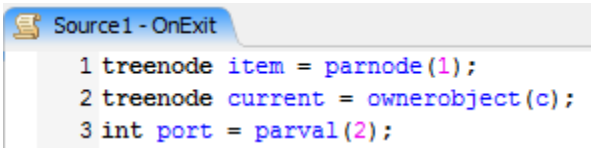
```
if(getitemtype(item)==1)

        return 3;

else return 5 + (getlabelnum(item, "nrofcopies")*(5.0/60.0));
```

As before we use an if statement to give itemtype 1 (package customers) a service time of 3 minutes. In the else portion (copy customers), though, we return the expression: 5 + (getlabelnum(item, "nrofcopies")*(5.0/60.0)). This is the base service time of 5 minutes plus the number of copies that the customer needs (getlabelnum(item, "nrofcopies")) times five seconds (5.0/60.0). Remember that we have defined our model in minutes, so if one FlexSim time unit is equal to one minute, then five seconds is equal to 5/60 minutes or time units.

Note on the division operator: In the above example I use the expression 5.0/60.0 instead of 5/60. It is important to make this distinction because C++ sees the two division expressions differently. With 5/60, C++ sees this as the integer 5 divided by the integer 60. Thus, an integer divided by an integer must also be an integer, or 0. With 5.0/60.0, however, C++ sees it as the division of two floating point numbers, and thus the result is a fraction between 0 and 1. On the other hand, FlexScript, which is not strongly typed like C++, actually interprets the expression 5/60 as the division of 2 floating point numbers, meaning you would be fine using 5/60 in FlexScript. However, in the few situations such as this where FlexScript's implementation deviates from the C++ implementation, we encourage you to write code that is cross-compatible with FlexScript and C++, and thus the correct expression would be 5.0/60.0. For more information on integer vs. floating point division, refer to the topic on writing logic in FlexSim.

So again, just as with the itemtype value, we can use labels to store data on flowitems (or objects), and then we can access that data to make decisions in the model.

Item and Current



The terms **item** and **current** are two access variables that refer to objects in FlexSim. When you edit the code of a given trigger or picklist, you will always see at the top of the code one or more "header" statements. These statements set up your access variables for you, and usually will look something like the following:

```
treenode item = parnode(1);

treenode current = ownerobject(c);

int port = parval(2);
```

For more information on port, see the Ports page.

Example

In this example, the first statement is what we call a variable declaration. For example, the second line declares a variable called **current**. The variable type of current is a treenode. This is a variable type that holds a reference to an object in FlexSim's tree structure. I don't want to go into too much detail on this, so in a nutshell, all data in FlexSim, including objects and flowitems, is stored as nodes in a tree structure, and the treenode variable type is simply a reference to a node (or object) in that tree structure. For more information on the tree structure, refer to the topic on FlexSim's tree structure.

The first statement's declaration also sets the value of this variable named current to: **ownerobject(c)**. Now, I also don't want to go into too much detail on what the meaning of ownerobject(c) is because that can be a complicated side-track. The essential thing here is that you have a treenode variable (or object reference) called **current**, and you'll just have to trust me when I tell you that current will always point to the "current" object that you are editing the field for. If you go into a Source's parameters window and edit the Source's exit trigger, then in that field, current is a reference to that Source object. If, on the other hand, you go into a Processor object's parameters window, and open the code for the Processor's process time field, then within that field, current is a reference to that Processor object.

The example code also has a second statement. The statement is another declaration of a treenode variable, called **item** this time, that is given the value: **parnode(1)**. Again, in order not to get side-tracked, I'm not going to explain the parnode command but will just say that item will always refer to the flowitem that is associated with a specific execution of that field or trigger. For example, if you are implementing the exit trigger of a Source, then each time the exit trigger is fired, item will refer to the flowitem that is exiting the Source at that specific time. Note that the item reference will change each time the exit trigger is executed because a new flowitem is exiting, whereas the current reference will be the same each time because the Source object does not change.

So these header statements set up the access variables that can be used within the code of the field. This is why in the previous examples I was able to use the word item in writing the commands:

```
setitemtype(item, bernoulli(60,1,2));
```

or:

```
if(getitemtype(item)==2)...
```

because I have a reference to the flowitem and the reference is named item.

Often the header statements mentioned above will vary depending on the type of field you are writing code for. For example, the header statements of an exit trigger should look like this:

```
treenode current = ownerobject(c);


treenode item = parnode(1);


int port = parval(2);
```

Here there is an additional variable declaration of an integer called **port**. In this case port is the output port number through which the item is exiting. A reset trigger's header statements, on the other hand, will look like this:

```
treenode current = ownerobject(c);
```

Here there is only one variable declaration, namely current. There is no item declaration. The reason for this is because the reset trigger, which is executed when you press the model reset button, has no specific flowitem associated with its execution.

This user manual documents each field and its access variables in the topics and sub-topics of the picklist section.

Review

So to review, within FlexSim's code fields you will often have access to variables called current and item. Current will always reference the object whose code you are editing. Item will always reference a flowitem that is associated with a specific execution of the field (e.g. the item that is exiting the Source). The access variables will vary based on the type of field, but you can always find which variables are available just by looking at the header statements at the top of your code, or by referring to the user manual.

Note on specifying the correct object when accessing labels or itemtype: It is important to understand which object is holding a label or itemtype. For example, in the example above we use the command *getlabelnum(item, "nrofcopies")*. We do not use *getlabelnum(current, "nrofcopies")*. The reason we use item and not current is because the label is stored on the flowitem itself, and not on the FlexSim object. If you add a label to a flowitem in the flowitem bin, then item should be the reference for the getlabelnum command. On the other hand, if you are using a label on the object (you have added the label to the object through its properties window), then current should be the reference in the getlabelnum command.

Return Values

In FlexSim there is a close interaction between the "under-the-hood" behavior of the objects in your model and the logic that you implement on those objects through code fields. Often the very reason that a code field is executed is because the FlexSim object (Processor, Source, etc.) is requesting data from you the modeller as to how it should operate. For example, the process time field of a Processor is executed because the Processor needs to know from you what its process time should be for a given flowitem. The way that you pass the correct information back to the Processor is through the return value of that field, or in other words, by executing a return statement in your code.

Example

Let's refer back to the post office example mentioned previously. In that example we implemented the process time code as:

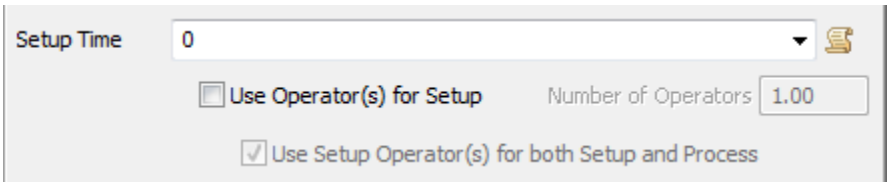
```
if(getitemtype(item)==1)

  return 3;

else

  return 5 + (getlabelnum(item, "nrofcopies")*(5.0/60.0));
```

The Processor will execute this field each time it receives a flowitem, just before it starts the process time. By executing this process time field, it is essentially asking you what the process time for that item should be. In this code we are using the **return** statement to pass back to the Processor the appropriate process time for that flowitem. Thus, the return statement is used to give back to an object the appropriate data that it needs to operate as we want it to.

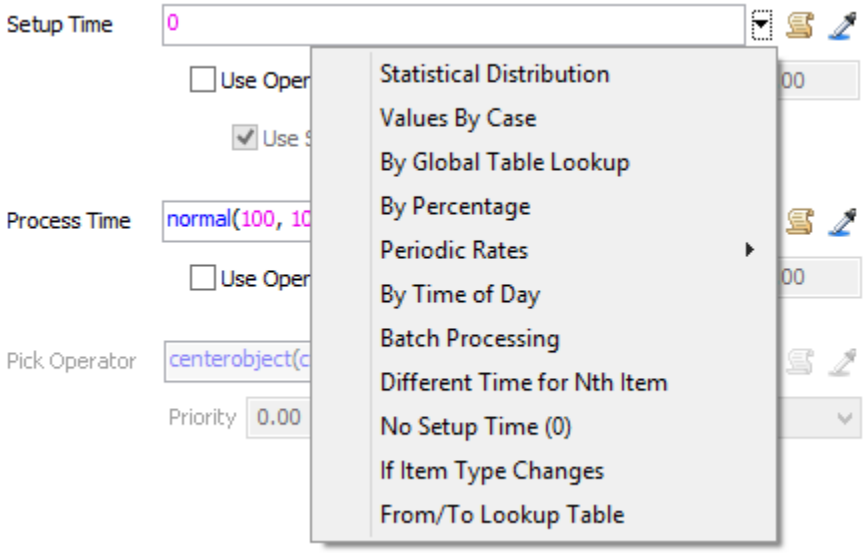
Many fields do not need a returned value. For example, in the post office model the Source's exit trigger does not include a return statement. The reason for this is because with an exit trigger, the Source object is not trying to get information back from you, it is simply providing you with a spot where you can execute functionality when a flowitem exits the Source.

This user manual documents each field's required return value in the sub-topics of the pick lists section.

Picklists

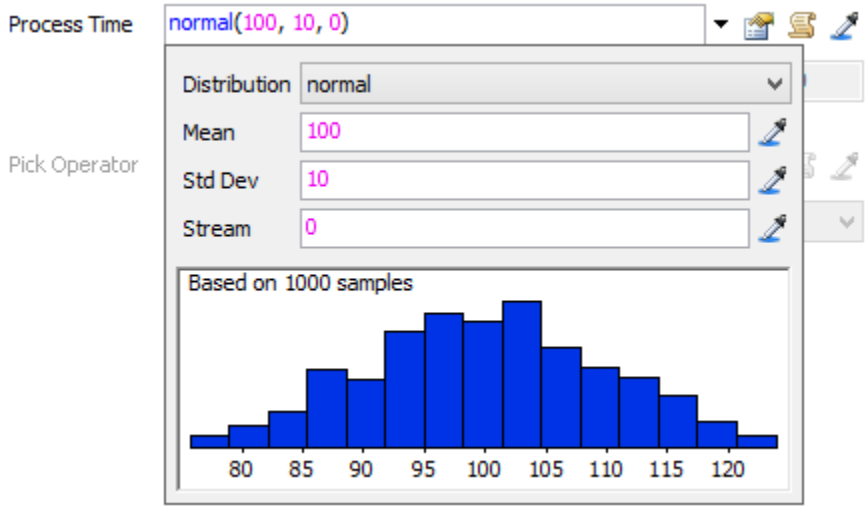


You will find picklist windows throughout FlexSim. These windows give you an easy interface for implementing functionality in FlexSim. Behind the scenes, each of these windows refers to a piece of code. The nice thing about these picklist interfaces is that they allow you to write functionality without writing code. They give you a list of commonly used functionality when you click on the drop-down box. You can also select the text and replace it with a constant or an expression.



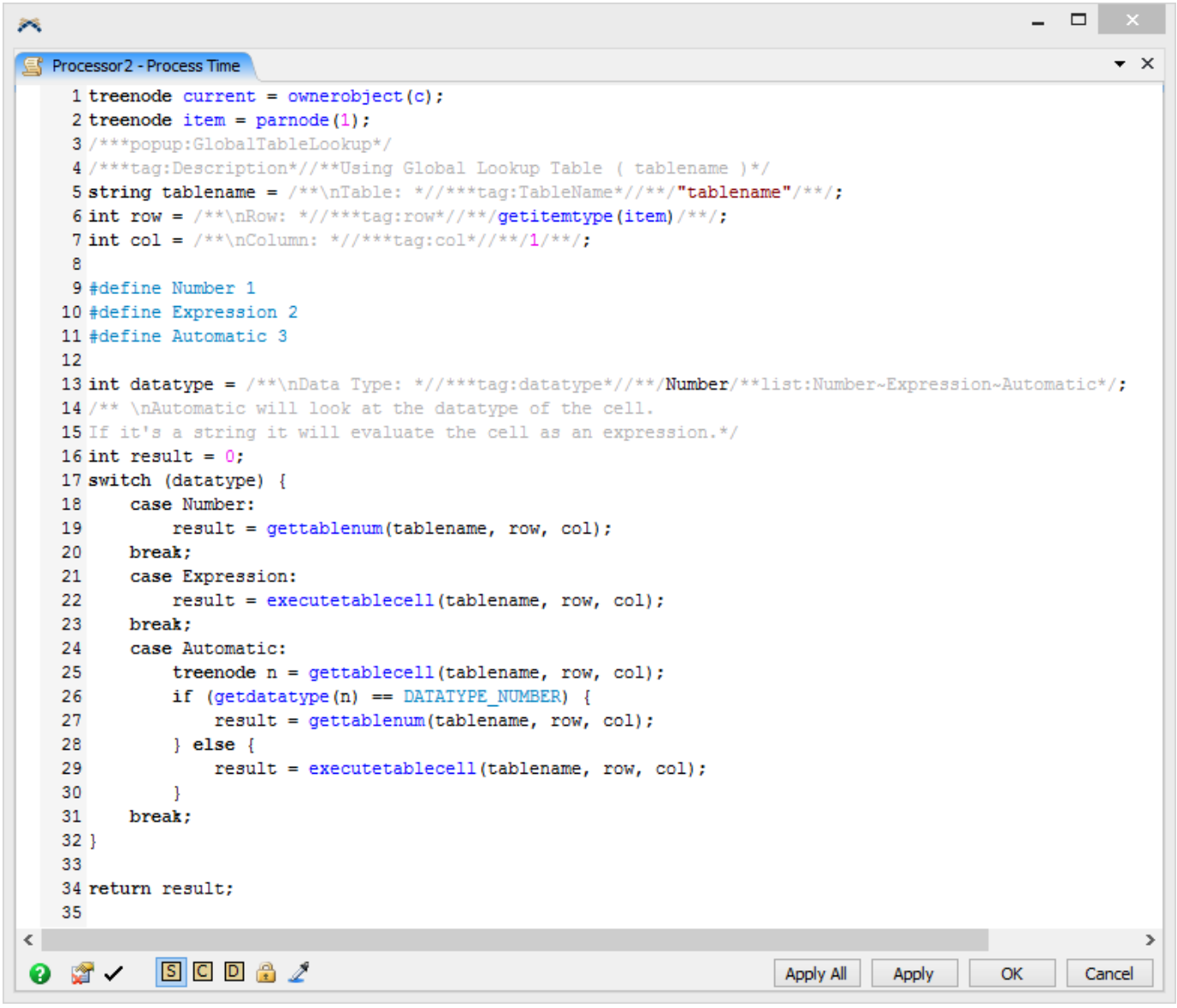
Popups

Each picklist option has an associated popups for user interface. Popups allow you to easily edit the option parameters. You can edit these options at any time by clicking the button. Once you have entered the values you want in the popup, click anywhere outside the popup to close it.



Code Edit

Experienced modelers also have the ability to write the code explicitly when needed. By clicking on the code edit button , you can bring up the code edit window, in which you can see all of the code that implements this field. Note that much of the code you will see is actually used to format the code template window. You can decipher the real code from the code template formatting code by the color. Code template formatting code is commented out in gray (see the Template Code page).


Processor2 - Process Time
1 treenode current = ownerobject(c);
2 treenode item = parnode(1);
3 /***popup:GlobalTableLookup*/
4 /***tag:Description*//**Using Global Lookup Table (tablename)*/
5 string tablename = /**\nTable: *//***tag:TableName*//**/"tablename"/**/;
6 int row = /**\nRow: *//***tag:row*//**/getitemtype(item)/**/;
7 int col = /**\nColumn: *//***tag:col*//**/1/**/;
8
9 #define Number 1
10 #define Expression 2
11 #define Automatic 3
12
13 int datatype = /**\nData Type: *//***tag:datatype*//**/Number/**list:Number~Expression~Automatic*/;
14 /** \nAutomatic will look at the datatype of the cell.
15 If it's a string it will evaluate the cell as an expression.*/
16 int result = 0;
17 switch (datatype) {
18 case Number:
19 result = gettablenum(tablename, row, col);
20 break;
21 case Expression:
22 result = executetablecell(tablename, row, col);
23 break;
24 case Automatic:
25 treenode n = gettablecell(tablename, row, col);
26 if (getdatatype(n) == DATATYPE_NUMBER) {
27 result = gettablenum(tablename, row, col);
28 } else {
29 result = executetablecell(tablename, row, col);
30 }
31 break;
32 }
33
34 return result;
35
Apply All
Apply
OK
Cancel

To learn more about the Code Editor, see the Code Editor page.

For more information on how to write code in FlexSim, refer to writing logic in FlexSim.

Template Code

In the code edit fields you may find seemingly weird gray text strewn throughout the code. For example, you might find the following piece of code:

```
/**By Expression*/
/** \nExpression: */
double value = /**/10/**/;
return value;
/** \n\nNote: The expression may be a constant
value or the result of a command (getitemtype(),
getlabelnum(), etc).*/
```

The gray text is called template code. Template code is used in picklist popups as talked about in the Picklists page. When the button is pressed, FlexSim parses the template code and displays parameters in the correct places within the popup.

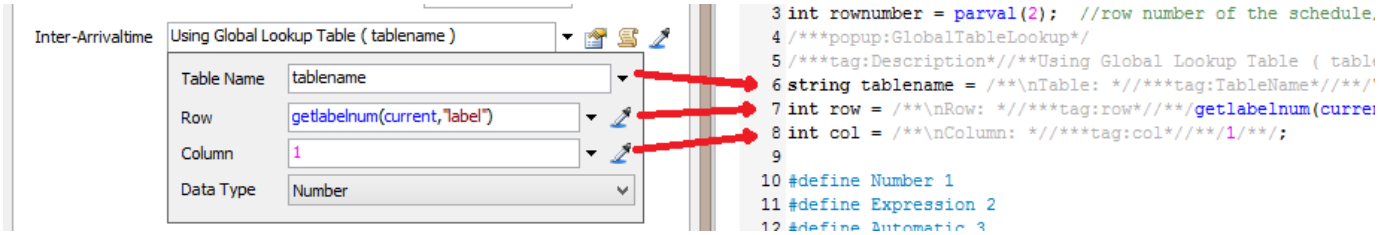
`/***popup:ValuesByCase:hasitem=1:valuestr=Port:doreturn=1*/`

This template code is an instruction to open a popup.

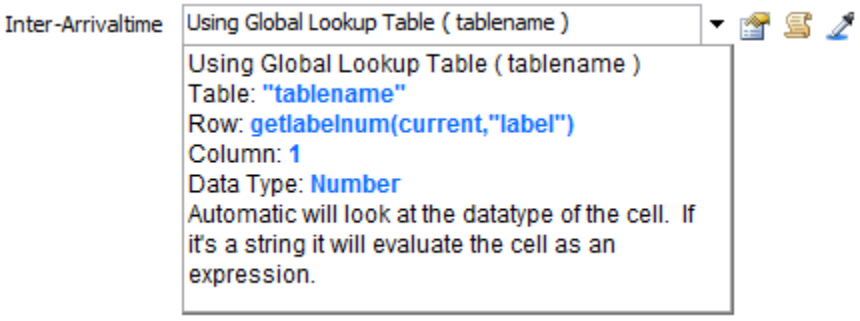
`int case_val = /***tag:ValueFunc*//**/getitemtype(item)/**/;`

When the popup opens, it parses the script for instances of `/***tag:TagName*//**/Value/**/` and places `Value` in the appropriate field in the popup. When the popup closes, the values in each field take the place of their respective `Value`. In this way, the `/***tag:` mechanism creates a two-way link between the script and the popup.

Here's a screenshot from from Using Global Lookup Table from the **Inter-Arrivaltime** drop-down menu in the **Properties** window of a **Source** object:



Template Text Popups


Inter-Arrivaltime
Using Global Lookup Table (tablename)
Using Global Lookup Table (tablename)
Table: "tablename"
Row: getlabelnum(current,"label")
Column: 1
Data Type: Number
Automatic will look at the datatype of the cell. If it's a string it will evaluate the cell as an expression.

If no popup exists for the template code, or if a user creates custom template code, FlexSim will display a template text popup. The above popup is editing the same values as the Popup displayed above. Template Text consists of black and blue text. Blue text is editable text, just like you would edit in an edit field of a normal popup. In your code, to specify a section of fixed black text, use a multi-line comment but add an additional asterisk to the start tag: /** . By adding this extra asterisk, it signals to FlexSim's template code

interpreter that this is a section of fixed black text that should show up when the user looks at the template text. In the example at the top of this section, the text: /**By Expression*/ makes it so the text "By Expression" will show up in black text in the template drop-down.

To specify the blue editable text, you want the input from the user to be part of the actual code. So to start blue text, you go into a multi-line comment then immediately go out of the comment: /**/ . You use the same tag to get out of a section of blue text. Thus, in the code above, the value 10 is made available for changing when the template text is shown:

```
double value = /**/10/**/;
```

Because these are comments, the FlexScript parser only sees: double value = 10; but the advantage is that now you (or another modeller) can quickly change the 10 value to something else just by pressing the button and editing the blue template text.

You may also notice that the comments will occasionally include a \n tag. This tag specifies a new line to be made in the fixed black template text. You can also specify a new line just by putting a new line in the comment, but often you will want your template code to take up as little space as possible so that the code itself can be viewed more easily.

Comments

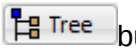
In FlexScript, you can "comment out" a section of text so that the FlexScript parser does not look at that text as part of the code. This allows you to add descriptive text that explains what the code does. There are two ways to make comments. The first is the one-line comment, and is done with two forward slashes: //. The example below shows a one-line comment

```
// This is my one-line comment, it ends at the end of this line
```

A one-line comment extends to the end of that line of text. The other comment is a multi-line comment. Here you signal the start of the comment with the text: /* and signal the end of the multi-line command with the text: */ as shown below:

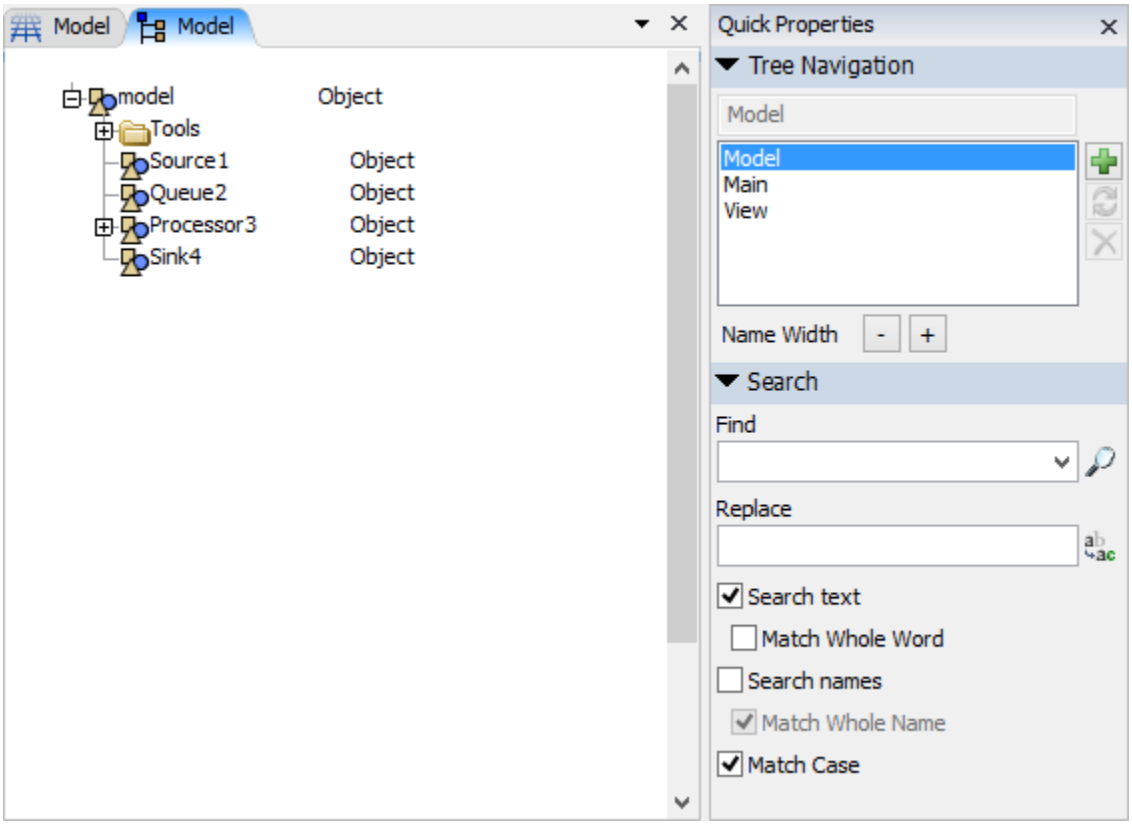
```
/*
this is my multi-line comment
it can span as many lines
as I want it to
*/
```

Model Tree View

The **Model Tree** view is used in FlexSim to explore the model structure and objects in detail. To access the model tree view select the 

button from the toolbar. The Tree Window will then appear and the Quick Properties window will change to display the Tree Navigation and Search sections.

Note: If you are using the Evaluation version of FlexSim, you will not be able to use the Model Tree View.



The model tree view is a view window that provides many unique features. In this view you can:

- Customize FlexSim objects using C++ or FlexScript
- View all object data
- Access the properties windows
- Edit the model, delete objects, and modify all data

If you follow a few simple navigation rules you will find the tree view to be one of the most versatile views within FlexSim. The underlying data structure in FlexSim is contained in a tree. The many edit windows within FlexSim are simply graphical user interfaces (GUIs) that display filtered data from the tree. Since all tree views in FlexSim work the same way, once you understand how the tree view works you will be able to navigate and understand the structure of any tree view that is accessible.

Tree View Basics

FlexSim has been designed to hold all data and information in a tree structure. This tree structure is the core data structure for all of FlexSim's object oriented design. Those who are familiar with C++ object oriented programming will immediately recognize FlexSim's tree view as the C++ standard for object oriented data management.

There are several symbols used in the tree view that will help you understand the structure as you navigate the tree.

The entire MAIN tree view is referred to as a project. The library and model are contained in a project. The VIEW tree contains all the views and GUI definitions as well as all the currently open windows. When a session is saved the MAIN tree and the VIEW tree are saved together.

The folder icon identifies a node that does not have any object data, but may contain other folders or objects.

The object icon is used to represent FlexSim objects in the tree view.

The node icon is used to specify data nodes within an object. Data nodes can have additional data nodes placed inside them. If a data node has a "+" just to the left of the icon it will contain one or more additional data nodes. Data nodes can hold numeric or alphanumeric values.

Certain data nodes are used to hold executable code. There are four types of code nodes in FlexSim:

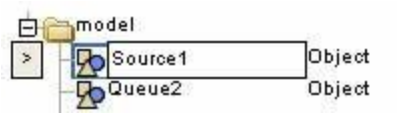
C++. C++ code must be compiled before running the model.

Flexscipt. This code will be auto-compiled during the running of the model. Read more details on writing logic in FlexSim.

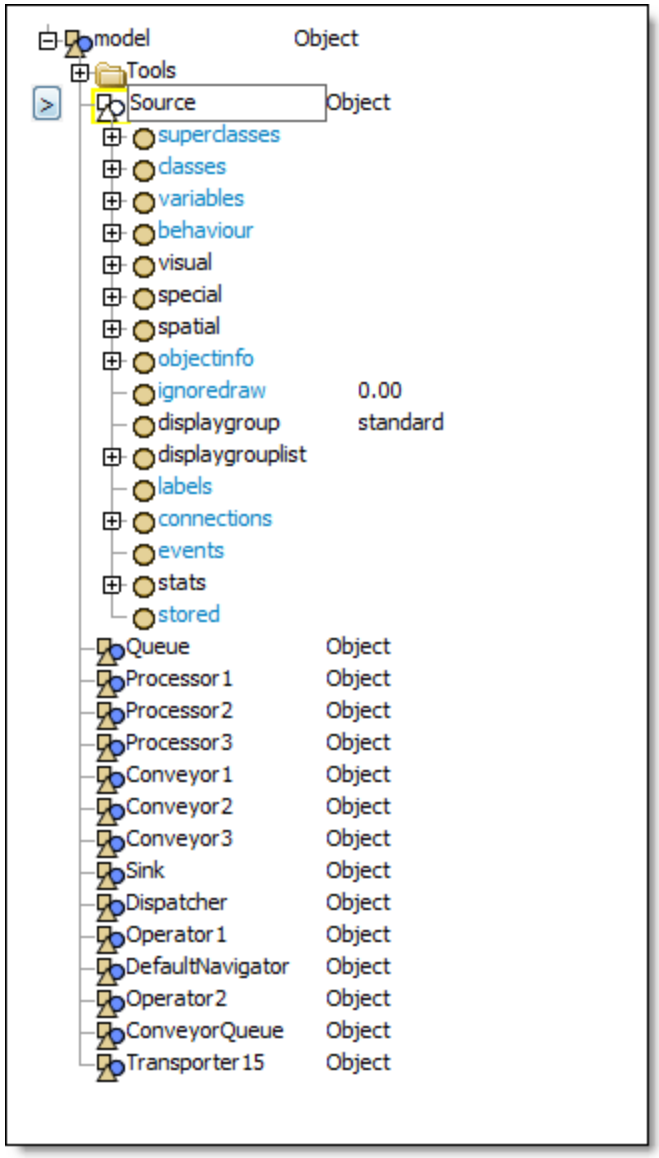
Dll. This node refers to a FlexSim compatible pre-compiled dll. To create such a dll you would need to use a special Visual C++ project. This project is available on the user community.

Global C++. This is C++ code that is globally scoped. It must contain complete functions. These functions can be accessed by any node nested below this node. Typically this type of node is found as the first node under the main Tool folder.

When you select an object in the tree view by clicking on the icon with the mouse, the tree view will display the object as follows:



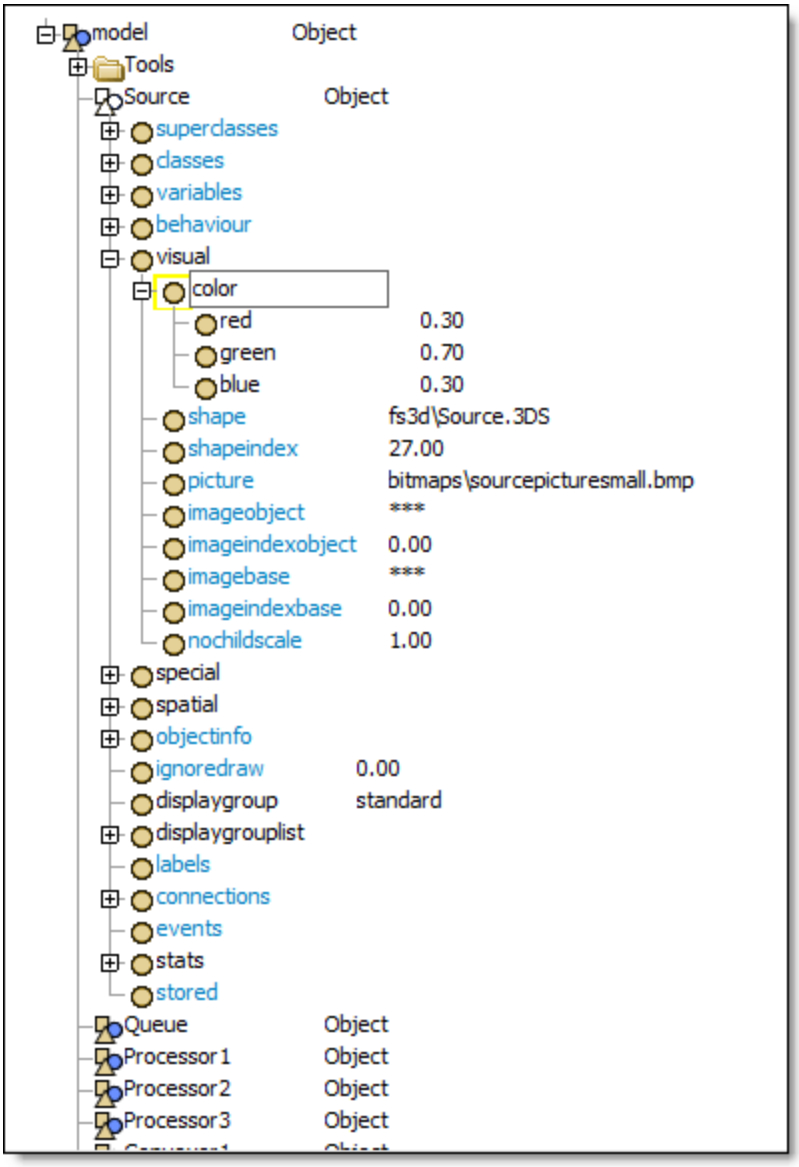
A highlighting box will be placed around the object icon and an expand tree symbol will be place to the left of the object icon. If you select this expand tree symbol, the data nodes for that object will be displayed as shown below.



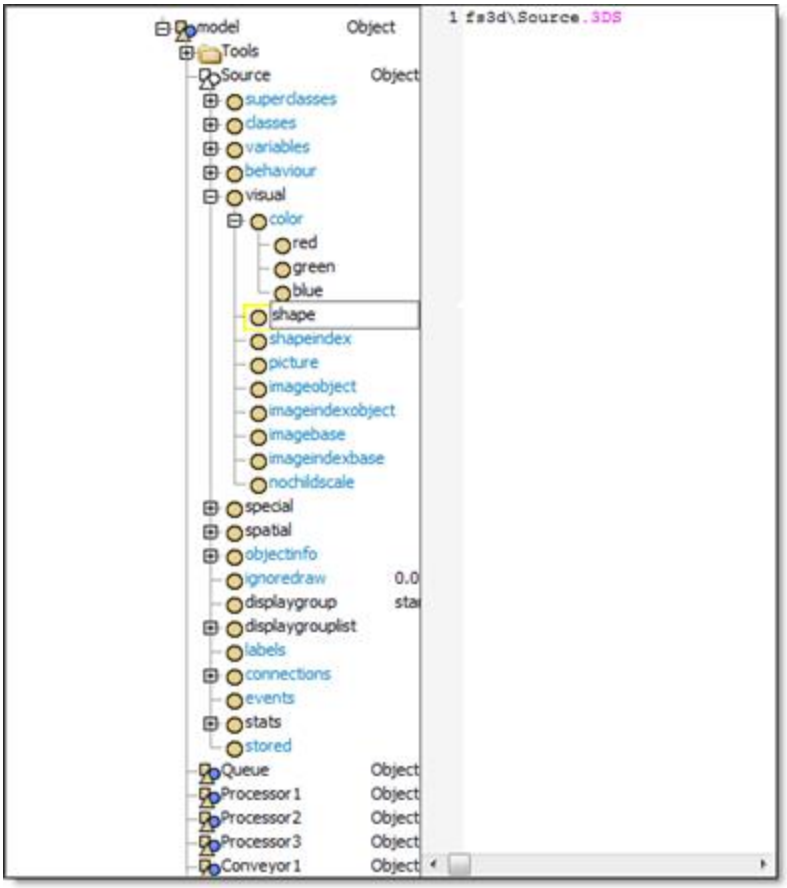
As objects and data nodes are expanded, the tree view can quickly grow to be outside the viewing limits of the tree view window. FlexSim allows you to move the tree around in the window by using the mouse. To move the tree around in the window just click-and-drag on the left side of the tree, use the mouse wheel to scroll up and down, or use the scroll bar on the side.

Data nodes can be expanded by clicking on the "+" to the left of the node icon. Since data nodes can have values or text you will see the text information or the data values to the right of the node.

If you select on an object or data node you may not be able to move the tree. Click a spot in the view that is blank, then drag the mouse to move the tree up and down. You can also use the scroll bar, mouse wheel, or PageUp/PageDown buttons to move the tree up and down.



Data can be edited directly in the tree by selecting the node you wish to edit. If it is a numeric data node you will be able to edit the number in the field. If it is a text data node you will be given a text edit field on the right side of the window to edit the text.



As you can see, the tree is the repository of all data for the model. The properties windows are used to provide a more user-friendly way to manipulate the data in the tree. It is possible to completely edit your model from the tree, but it is recommended that you use the properties windows to avoid inadvertent

deletion of model data. The properties windows are accessible in a tree view by double-clicking on the object icon or by selecting Properties from the right-click menu.

For more information on the Tree view, see the Tree Window page.

Tutorials

1. Introduction
2. Lesson 1
3. Lesson 2
4. Lesson 2 Extra Mile
5. Lesson 3
6. Labels
7. Global Modeling Tools
8. User Events
9. Time Tables
10. Kinematics
11. Task Sequence Tutorials
    - Tutorial 1
    - Tutorial 2
    - Tutorial 3
12. SQL Tutorial
13. Fluid Objects
14. Experimenter / Optimizer Tutorial

Tutorials Introduction

This basic tutorial will take you through the steps of setting up a process flow, building a model, inputting data, viewing the animation, and analyzing output. Each lesson will build upon the previous one. It is therefore important to thoroughly understand each lesson before moving on to the next one. You should plan on at least 45 minutes to complete each lesson. Lesson 2 will also include an "Extra Mile" section at the end that will add additional value to your model.

Each lesson will have the following format:

1. Introduction
2. What you will learn
3. Approximate completion time
4. Model description
5. Model data
6. Step-by-step model construction

The following lessons are contained in this tutorial:

Lesson 1

Building a simple model that will process 3 different flowitem types. Each itemtype will have a specific routing. Objects used in this model will be the Source, Queue, Processor, Conveyor, and Sink. The basic statistics of model performance will be introduced, and the parameter options for each object will be explained.

Lesson 2

Using the model from lesson 1, you will add Operators and Transporters to the process. Object properties will be introduced, and additional statistical analysis concepts will be discussed.

Lesson 2 Extra Mile

After you have completed lesson 2 you will be shown how to add 3D charts and graphs to the model using **Dashboards**. 3D visual text will also be added using the VisualTool object to give annotation to the model.

Lesson 3

Using the model from lesson 2, you will add rack storage and network paths. Advanced statistics and model logic will be added, as will global tables used for reading and writing data.

Labels

This lesson will introduce you to the use of labels. You will also be introduced to Pull Requirements.

Global Properties

This lesson will introduce you to the basic concepts of global properties that can be used throughout your model. You will also be introduced to the Combiner and Separator objects.

User Events

This lesson will introduce you to User Events. Which can be used to execute specific code at specified times during the model run.

Time Tables

This lesson will introduce you to Time Tables.

Kinematics

This lesson will introduce you to using Kinematics to perform simultaneous movements with a single object.

Task Sequences

This lesson will introduce you Task Sequences and creating custom Task Sequences.

SQL

This lesson will introduce you to reading from and writing to a SQL database from FlexSim.

Feel free to contact our technical support group if you have any questions while working on these tutorials. FlexSim technical support can be reached at 801-224-6914, or you can contact us through a support request by going to the Help Menu and selecting Support Request. We hope you enjoy learning how FlexSim can help you optimize your processes.

Note: The statistics in the models you build might not be exactly the same as those found in these tutorials. Small differences in simple things like object placement can change the long term results. For the purpose of these tutorials, these differences are not important.

Lesson 1 Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

Lesson 1 introduces the basic concepts of diagramming and building a simple model. Building a diagram of the process is a great way to start every model that you will build in FlexSim. If you can not build a diagram, flowchart, or at least see a picture in your mind of how the process works, you will have a difficult time building the model in FlexSim.

Note: if you have already gone through the Getting Started tutorial, many of the concepts you learn in this lesson will not be new. However, subsequent lessons build upon this lesson, so it is probably a good idea to go through it anyway.

What You Will Learn

- How to build a simple layout
- How to connect ports for routing flowitems
- How to detail and enter data into FlexSim objects
- How to navigate in the animation views
- How to view simple statistics on each FlexSim object

New Objects

In this lesson you will be introduced to the Source, Queue, Processor, Conveyor, and Sink objects.

Approximate Time to Complete this Lesson

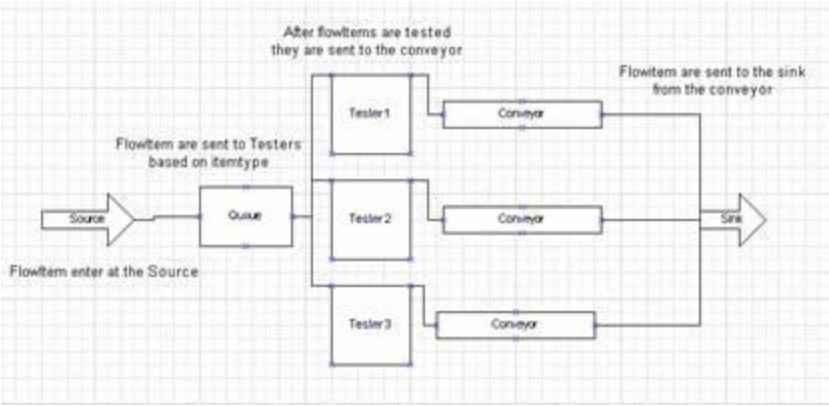
This lesson should take about 30-45 minutes to complete.

Model Views

FlexSim uses a three-dimensional modeling environment. There are two options for the view: perspective view and orthographic view. The orthographic view will look more flat, where as the perspective view will have a more real world feel to it. You may use any view option to build or run the model. You may open as many view windows as you want in FlexSim. Just remember that as more view windows are opened the demand on computer resources increases.

Model 1 Description

In our first model we will look at the process of testing three products coming off a manufacturing line. There are three different flowitem itemtypes that will arrive based on a normal distribution. Itemtypes will be uniformly distributed between itemtypes 1, 2, and 3. As flowitems arrive they will be placed in a queue and wait to be tested. Three testers will be available for testing. One tester will be used for itemtype 1, another for itemtype 2, and the third for itemtype 3. Once the flowitem is tested it will be placed on a conveyor. At the end of the conveyor the flowitem will be sent to a sink where it will exit the model. Figure 1-1 shows a diagram of the process.



Click here for the Step-By-Step Tutorial.

Model 1 Data

Source arrival rate: normal(20,2) seconds

Queue maximum size: 25 flowitems

Testing time: exponential(0,30) seconds

Conveyor speed: 1 meter per second

Flowitem routing: Itemtype 1 to Tester 1, Itemtype 2 to Tester 2, Itemtype 3 to Tester 3.

Step-By-Step Model Construction

Building Lesson 1 Model

Open the application by double clicking on the FlexSim icon on your desktop. Once the software loads, you should see the FlexSim menu and toolbars, Library, and Orthographic Model View windows.

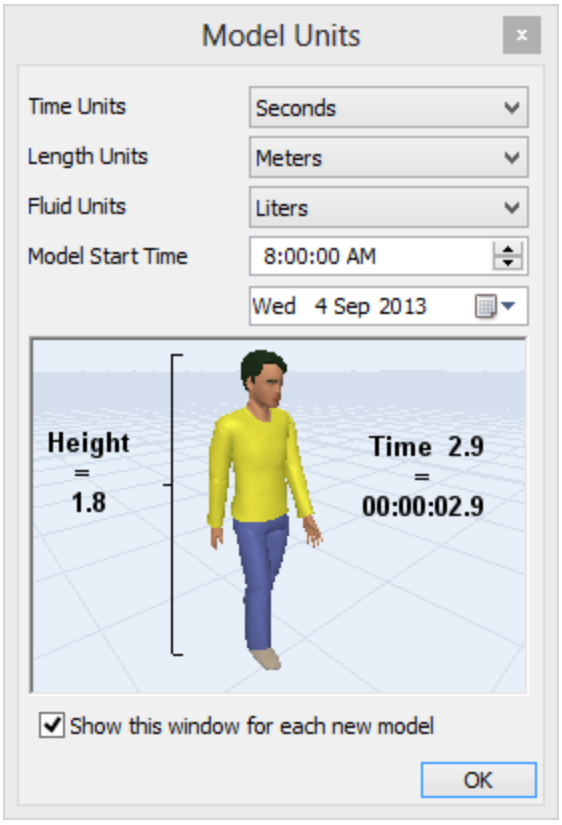
If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Starting a New Model

- Open FlexSim by double-clicking on the FlexSim icon on your desktop. The Startup Wizard appears by default. Select the "Build a New Model" option.

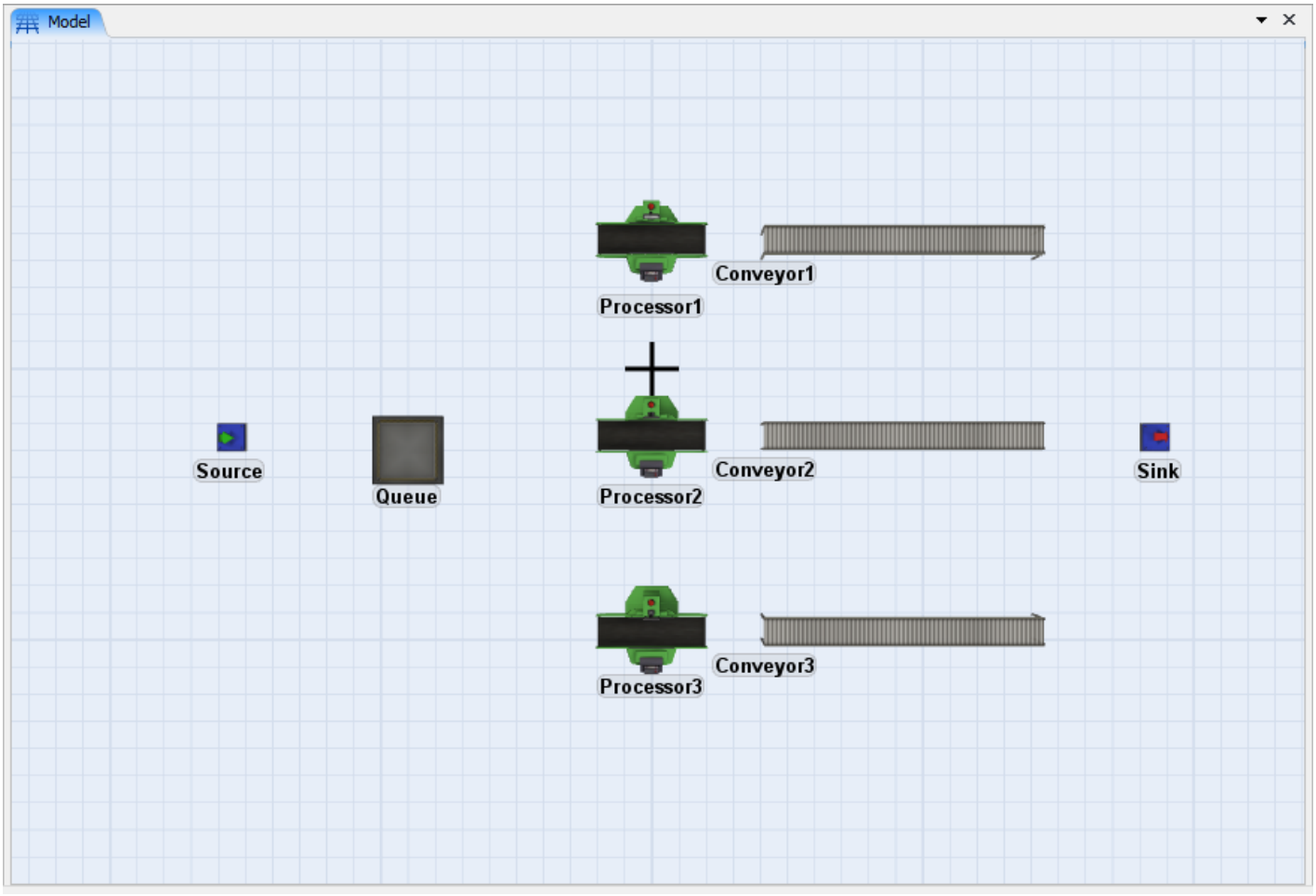
FlexSim allows the user to select appropriate units for a model. By default, the Model Units dialog will appear for each new model. You can select units for time, length, and fluids. The units you choose will be assumed throughout the model. For this model, use the following:

- **Time Units:** Seconds.
- **Length Units:** Meters.
- **Fluid Units:** Liters.



Step 1: Create the Objects

- Create a **Source** in the model and name it *Source* (To see how this is done, click here).
- Create a **Queue**, 3 **Processors**, 3 **Conveyors**, and 1 **Sink** in the model. Place and name them as shown below. To name an object: double-click on it, change its name at the top of the **Properties** window, and press **Apply** or **OK**. Click Here to see how this is done.

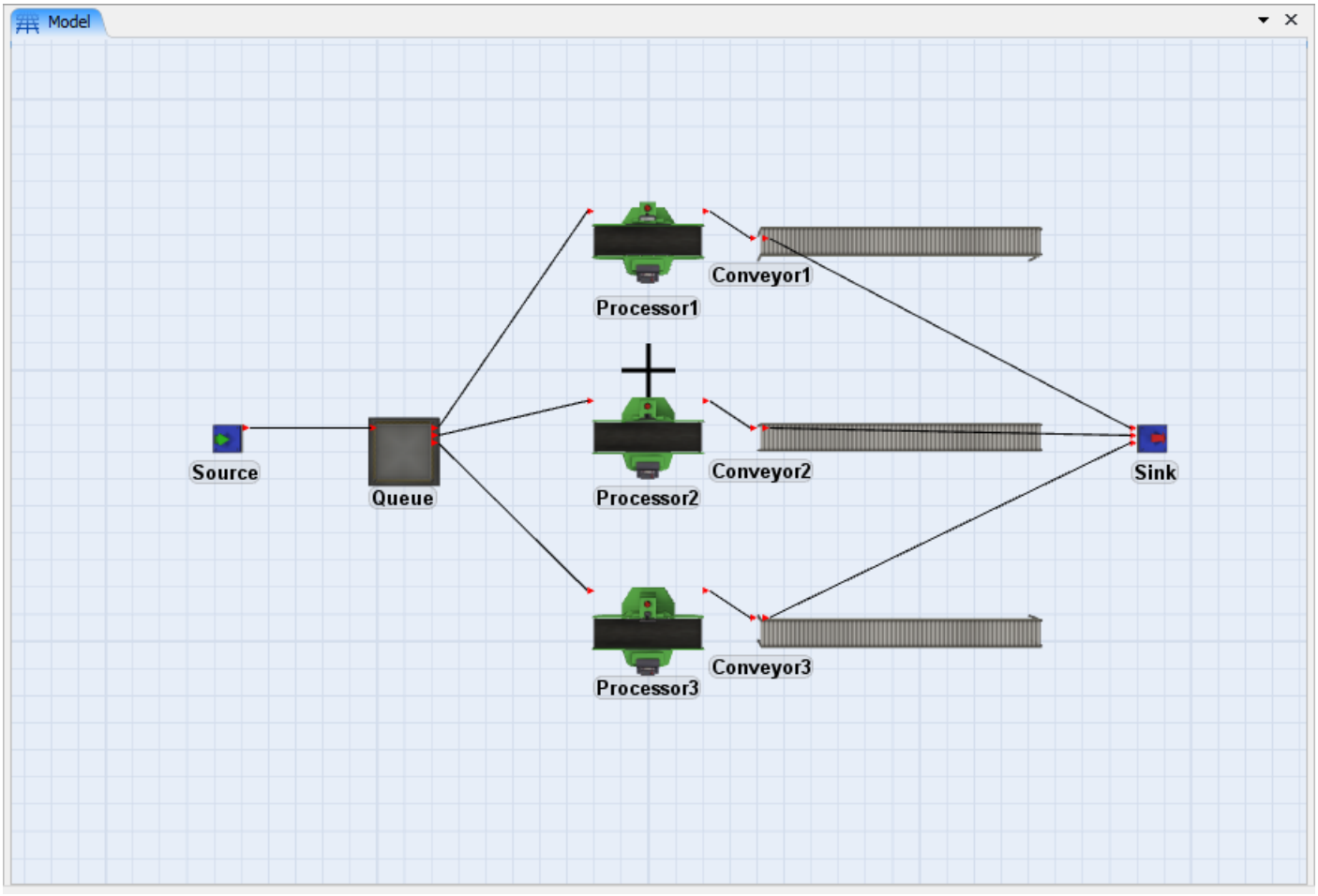


Step 2: Connect the ports

Enter the connection mode by either clicking the button or by pressing and holding the A key on the keyboard. Once in the connection mode, there are two ways to make a connection between objects. You can either click on one object and then click on another object, or you can click and drag from one object to the next. Either way, keep in mind that the flow direction of a connection is dependent on the order in which you make the connection. Flow goes from the first object to the second object in the connection.

Incidentally, connections can be broken by clicking the button or by pressing and holding the Q key on the keyboard while clicking or dragging from one object to another in the same manner as when you connected them.

- Connect *Source* to *Queue.*
- Connect Queue to *Processor1*, *Processor2*, and *Processor3.*
- Connect *Proceesor1* to *Conveyor1*, *Processor2* to *Conveyor2*, and *Processor3* to *Conveyor3*.
- Connect *Conveyor1*, *Conveyor2*, and *Conveyor3* to *Sink*.

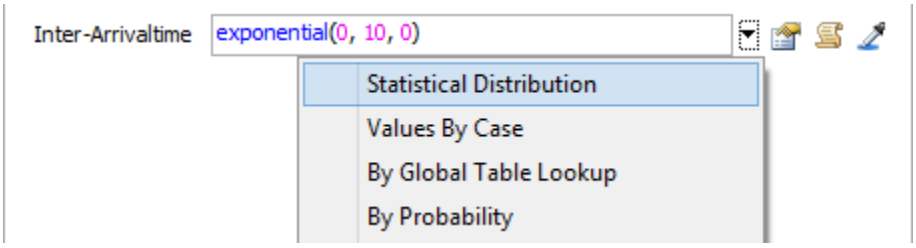


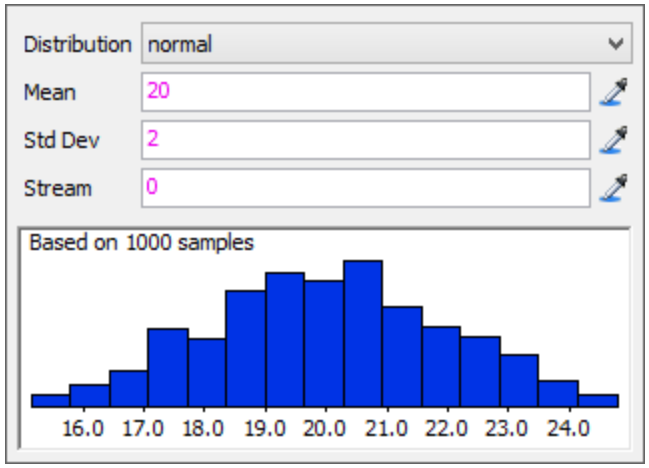
Step 3: Assign the arrival rate

In this model we need to change the Inter-Arrival time and the itemtype to generate 3 types of items.

- Double-click on the *Source* to open its **Properties** window
- On the **Source** tab, select **Statistical Distribution** from the **Inter-Arrivaltime** list. A statistical distribution popup will appear.
- Set **Distribution** to normal.
- Set **Mean** to 20.
- Set **Std Dev** to 2.
- Set **Stream** to 0.

This specifies an inter-arrival time with mean of 20 seconds and standard deviation of 2 seconds.





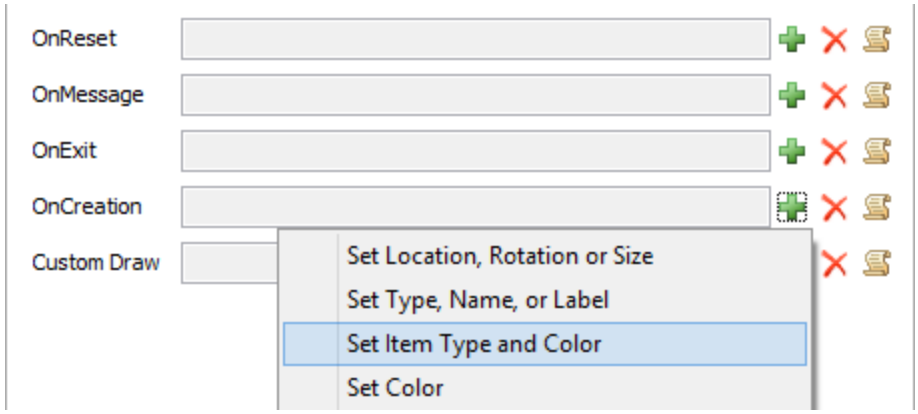
The next thing we need to do is assign an itemtype number to the flowitems as they enter the system. This will be uniformly distributed between 1 and 3. The best way to do this would be to change the itemtype on the OnCreation trigger of the Source, so don't close the Properties window yet.

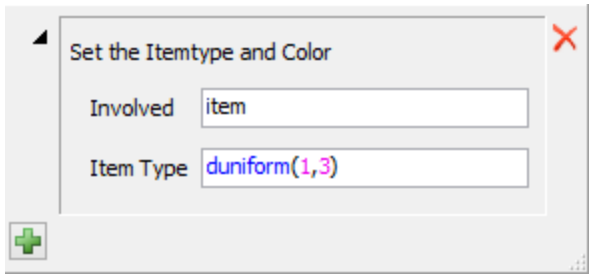
Step 4: Set Itemtype and Color

- Click the **Triggers** tab, and add a function (click the button) to the **OnCreation** trigger and select the **Set Item Type and Color** option. The code template popup will appear.

The duniform distribution is similar to a uniform distribution except that instead of returning any real number between the given parameters, only discrete integer values will be returned. The default values will be used in this example.

- Click **OK** to apply the changes and close the **Properties** window.





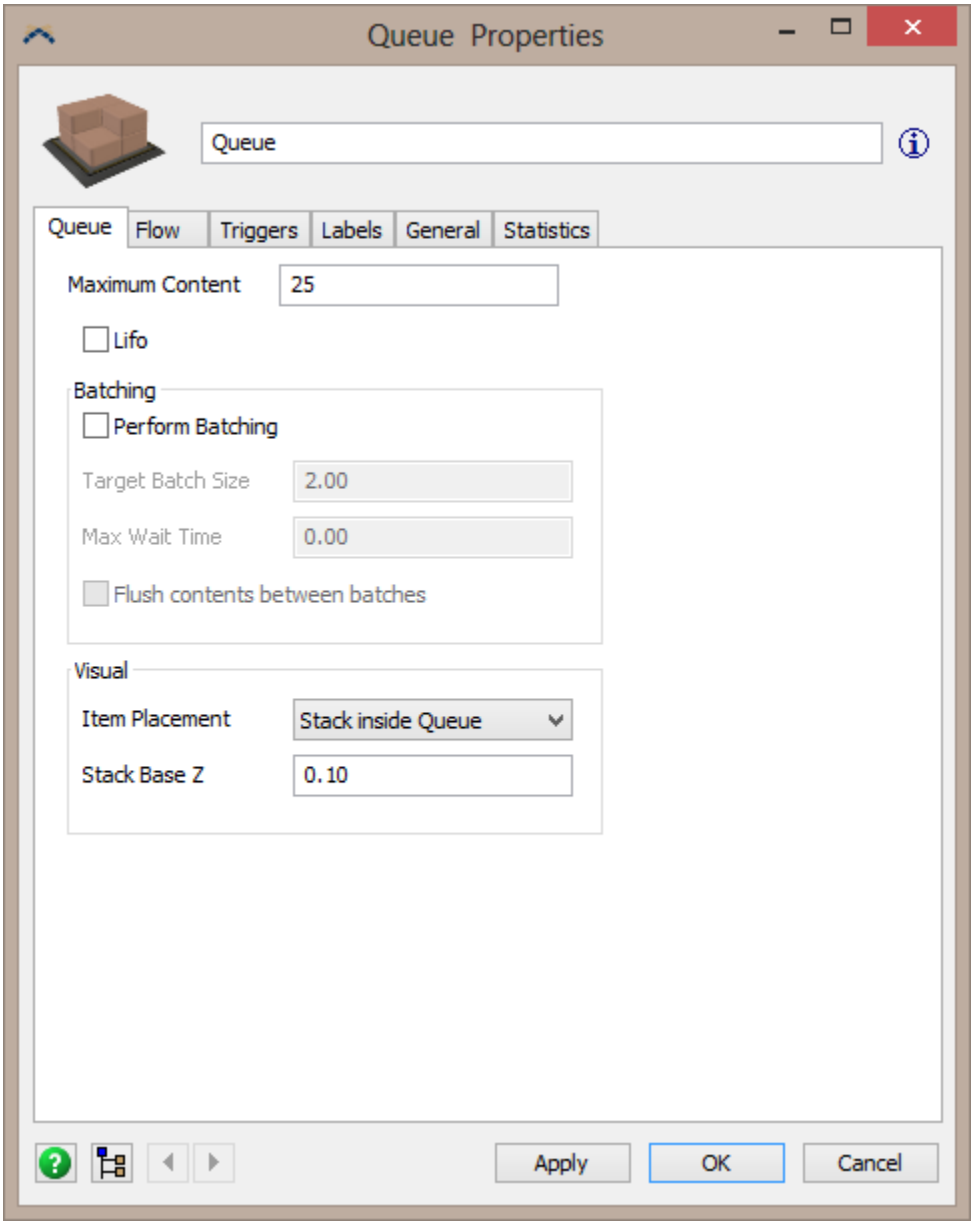
The next step will be to detail the queue. Since the queue is a place to hold flowitems until they can be processed at the processor, there are 2 things we need to do. First, we need to set the capacity of the queue to hold 25 flowitems. Second, set the flow options to send itemtype 1 to *Processor1*, itemtype 2 to *Processor2*, and itemtype 3 to *Processor3*.

Step 5: Setting the Queue Capacity

You may set the Queue's Maximum Content by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window:

- Double-click on the queue to open it's **Properties** window.
- Change the **Maximum Content** to 25.
- Don't close the Properties window yet.

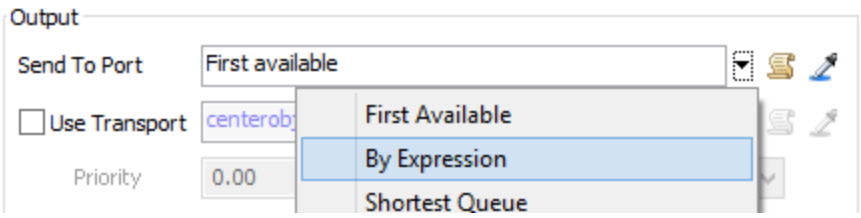


Step 6: Define the Flow for the Queue

You may define the Queue's flow by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window:

- Click the **Flow** tab in the Properties Window to set the flow options for the queue.
- On the **Send To Port** list, select **By Expression**



Since we have assigned an itemtype number equal to 1, 2, or 3, we can now use the itemtype to specify the port number through which flowitems will pass. Notice that the default output port is: getitemtype(item). Leave this as it is. Processor 1 should be connected to port 1, processor 2 should be connected to port 2 and so on. Click outside of the box to apply the trigger.

Click the **OK** button to apply and close the queue's properties window.

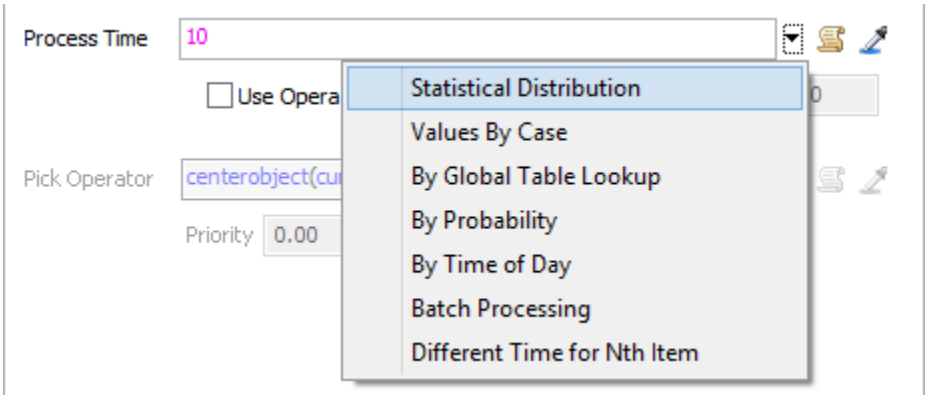
The next step is to set the processor times.

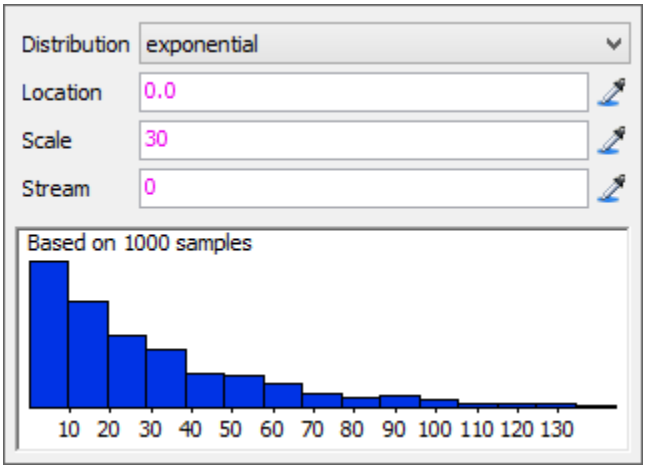
Step 7: Define the Process Time

You may define the Processor's Process Time by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window:

- Double-click *Processor1* to open its **Properties** window.
- On the **Processor** tab, in the **Process Time** section, select **Statistical Distribution** from the **Time** list. The statistical distribution popup will appear.
- Set **Distribution** to exponential.
- Set **Location** to 0
- Set **Scale** to 30.
- Set **Stream** to 0.
- Repeat this for the other 2 processors.





The default speed for the conveyors is already set to 1 length unit per time unit so there is no need to modify the conveyors at this time.

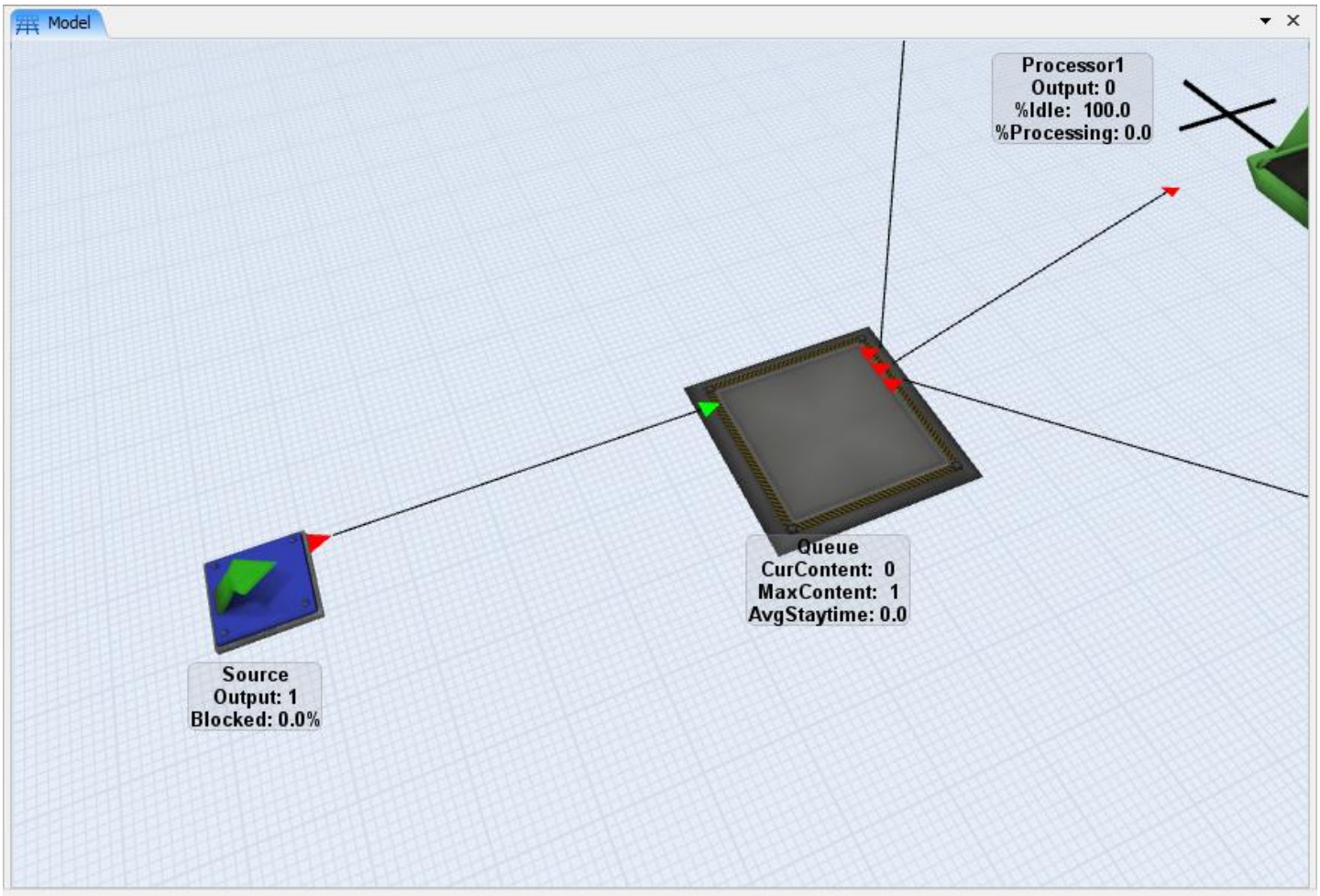
Now we are ready to run the model.

Step 8: Reset and Run the model

- Always click the Reset button to reset system and model parameters to their initial state before running a model.
- Click the Run button to start the simulation.

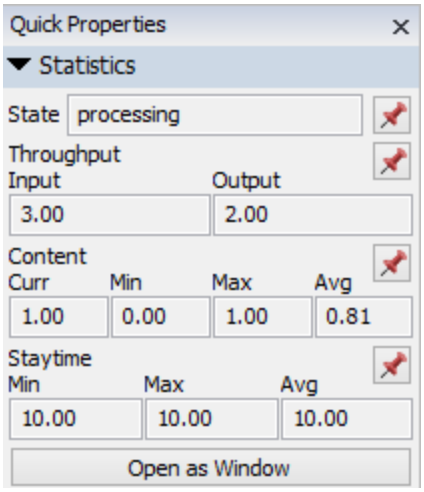
You should see flowitems entering the queue and then moving to the processors. From the processors, flowitems should move to the conveyors and then to the sink. You can change how fast the model runs by moving the Run Speed slide bar on the Simulation Run Panel.

Step 9: Viewing simple statistics



The above image shows how to view simple statistics for each object. If nothing is showing or only the names are showing, you can change the View Settings to show the statistics. To change the **View Settings**, click somewhere in the background of the view and in the Quick Properties window to the right, change the Show Names combo box to Show Names and Stats.

You can view more statistics of an object by clicking on the object and viewing the statistics tab in the Quick Properties.



Step 10: Save Model

Save your model by clicking the Save button on the main toolbar. The "Save FlexSim Model file" window will appear allowing you to navigate to the folder where you want to save your model. Change the "File name" to an appropriate name (lesson1.fsm) and select save. Remember that the file name extension must be .fsm.

You have now completed Lesson 1. Spend some time reviewing the steps and viewing the model as it runs. Congratulations!

To continue the tutorial, go to Lesson 2.

Lesson 2 Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

Lesson 2 introduces the concept of adding operators and transporters to a model, and explores object properties in greater detail. Lesson 2 also introduces additional graphical statistical output options. Make sure you have completed lesson 1 before starting lesson 2 as lesson 2 will use the model from lesson 1 as a starting point.

What You Will Learn

- How to access and modify object properties
- How to add a team of operators to the model
- How to add a fork truck transporter to the model
- How to view object statistics while the model is running

New Objects

In this lesson you will be introduced to the Dispatcher, Operator, and Transporter objects.

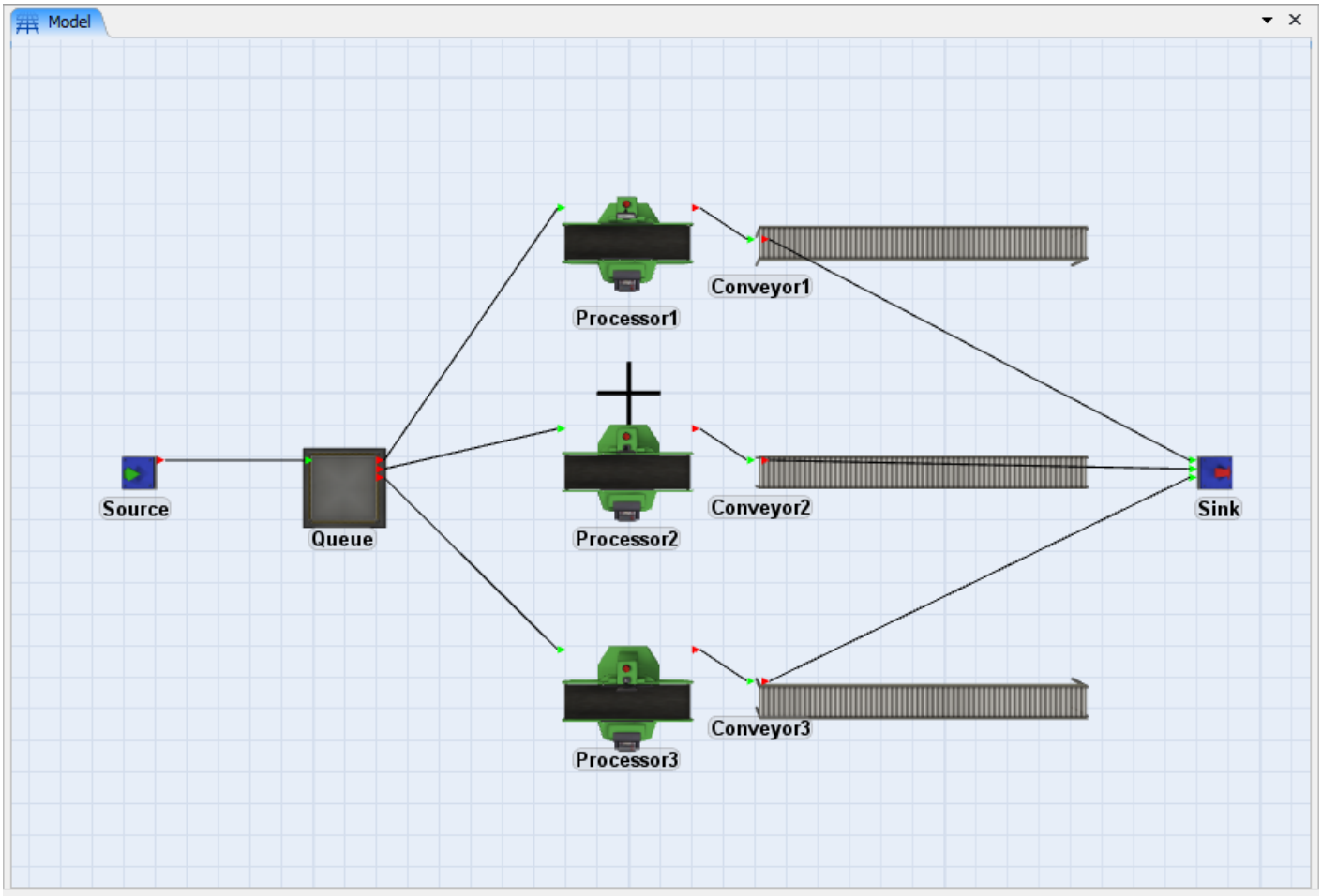
Approximate Time to Complete this Lesson

This lesson should take about 30-45 minutes to complete.

Model 2 Overview

In model 2 we will use a team of operators to perform the setup for the processing of the flowitems in the model. One of two operators will need to set up the processor. Once set up, the process can run without the need for the operator to be present. The operators must also carry the flowitem to the processor before the setup can begin. When the process is complete, the flowitem moves to the conveyor without the assistance of the operator.

When the flowitem reaches the end of the conveyor it will be placed in a queue where it will be picked up by a fork truck and taken to the sink. We may find it necessary to have more than one fork truck once we view the model as it runs. After the model is completed, view the default charts and graphs and address any bottlenecks or efficiency concerns. Below is a flow diagram of model 2.



Model 2 Data

Tester set-up time: Constant time of 10 seconds

Product handling: Operator from queue to tester. Fork truck from conveyor queue to sink.

Queue Capacity: 10

Step-By-Step Model Construction

Building Model 2

To start building model 2 you will need to load model 1 from the previous lesson.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

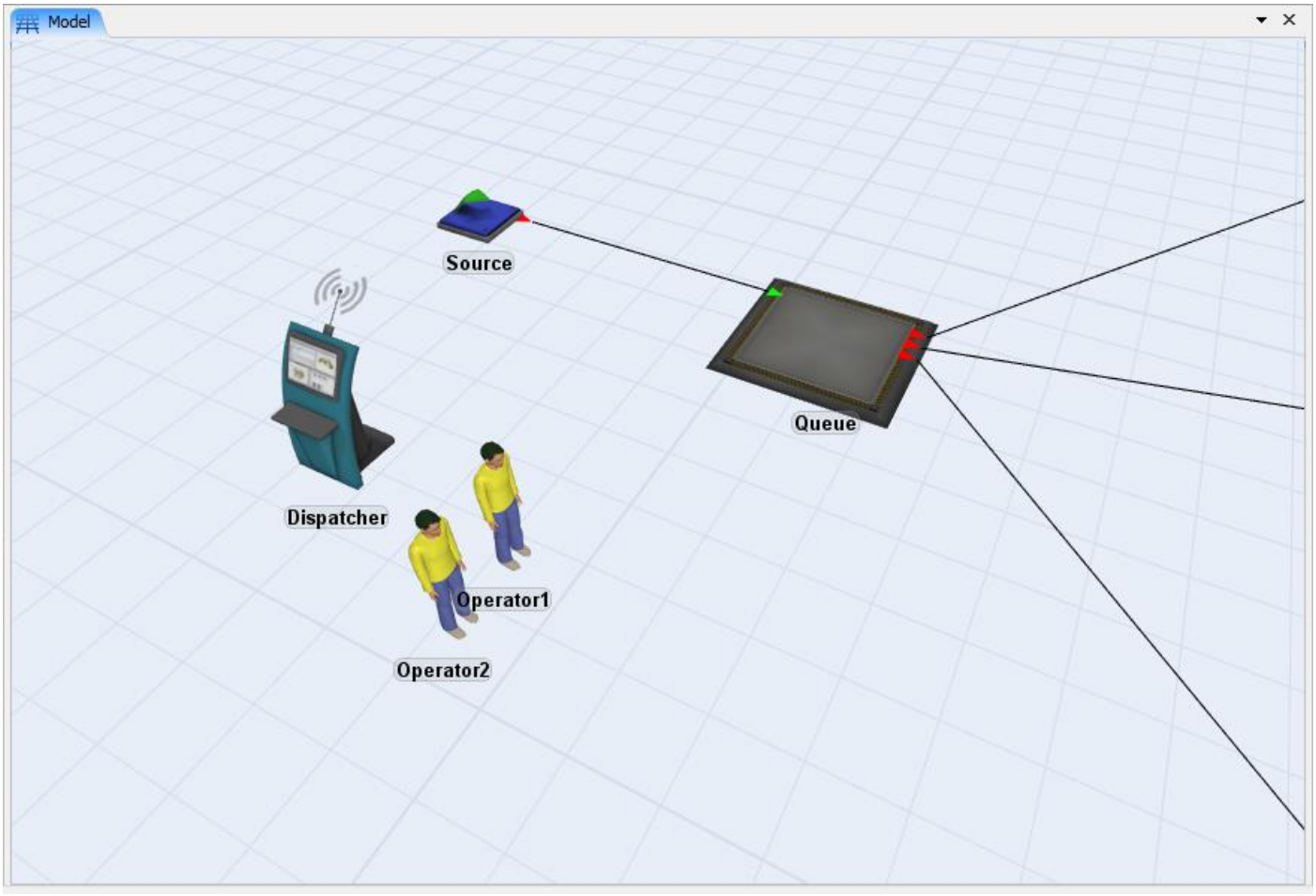
Step 1: Load model 1

- If you do not have Model 1 open, load it by clicking the button on the toolbar. Select the file for model 1 (.fsm file) saved from lesson 1.
- We want to make our flowitems exit the Source at a faster rate for this lesson. Double-click on the *Source* to open its **Properties** window, and under **Inter-Arrivaltime** change the **Mean** to 12.

Step 2: Create a dispatcher and 2 operators

The Dispatcher is used to queue up task sequences for a team of operators or transporters. In this case it will be used with 2 Operators that will be used to move flowitems from the queue to the testers.

- Create a **Dispatcher** and name it *Dispatcher.*
- Create 2 **Operators** and name them *Operator1* and *Operator2.*



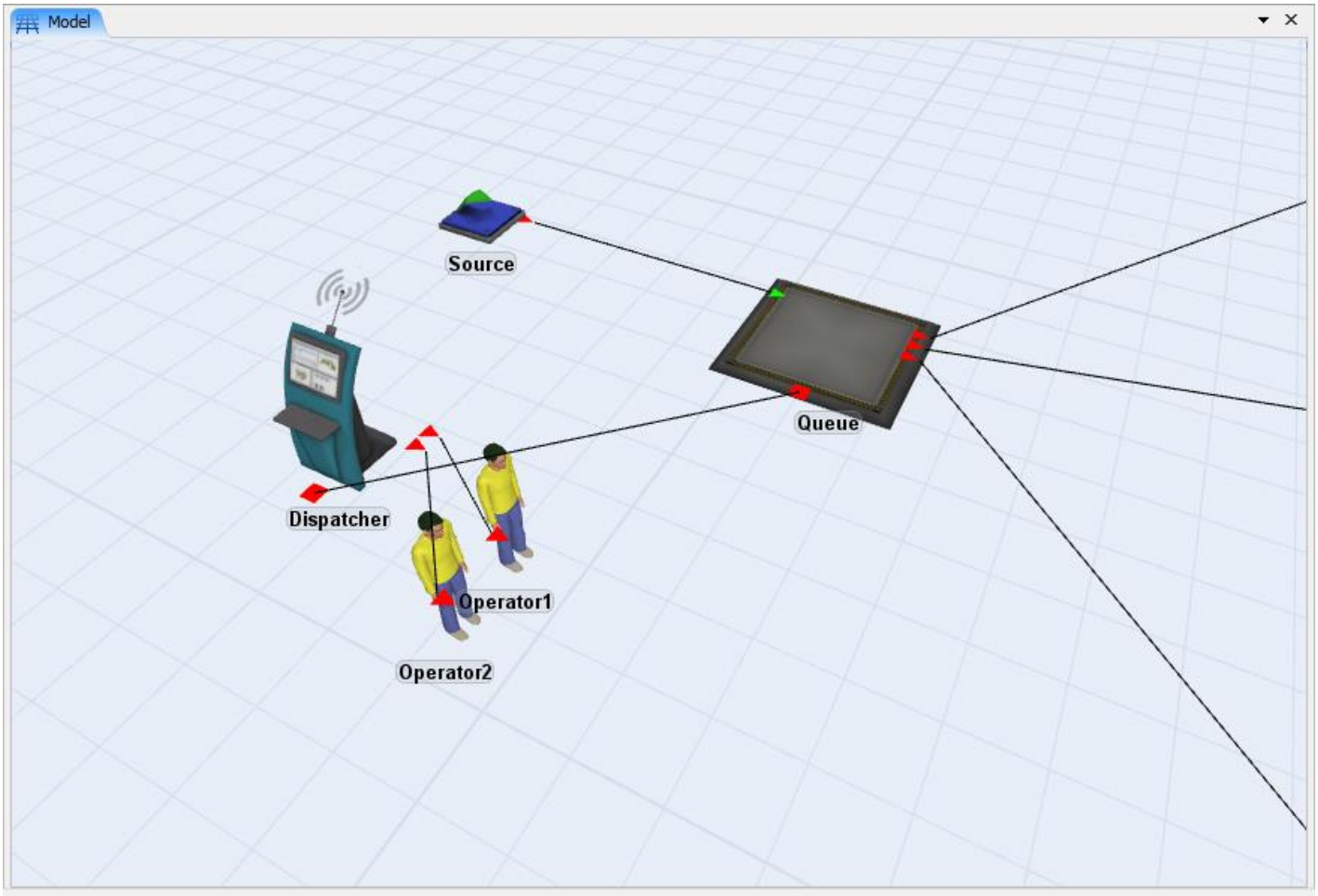
Step 3: Connect the Dispatcher and Operators

The queue is going to request an operator to pick up the flowitem and take it to one of the testers. The flow logic has already been set on the queue in lesson 1. You will not need to change the flow logic. You will only need to request an operator to be used for the task. Since we are using 2 operators, we will use a dispatcher to queue the requests and send them to a free operator to perform the task. If we only had 1 operator, we would not need the dispatcher and could connect the operator to the queue directly.

In order to use the dispatcher to direct a team of operators to perform a task, the dispatcher must be connected to the central port of the object requesting the operator. The central port is located in the center bottom position of the object. To connect the central port of the dispatcher to the queue, press and hold the "S" key on the keyboard and then click-and-drag from the dispatcher to the queue. When you release the mouse button a connection from the dispatcher's central port will connect to the central port of the queue. You can also use the Connect Center Ports mode from the Mode Toolbar to do this.

In order for the dispatcher to send tasks to the operators, the dispatcher's output ports needs to be connected to the operators' input ports. This must be done for each operator.

- Connect *Queue* to *Dispatcher* with a center connection (S key).
- Connect *Dispatcher* to *Operator1* and *Operator2* with a standard connection (A key).



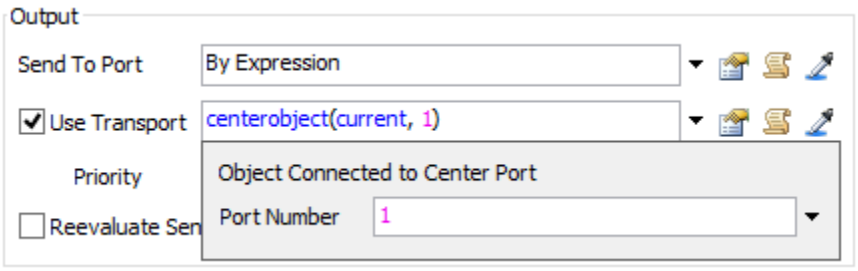
Step 4: Modify Queue Flow to Use Transport (Operators)

The next step is to modify the Queue's flow parameters so it uses the operators.

You may define the Queue's flow and transport options by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window.

- Double-click *Queue* to open its **Properties** window.
- Click the **Flow** tab
- Check the **Use Transport** box. The **Request Transport From** list will become available. This picklist allows you to select which Transporter or Operator to move the item based on the center port number. In this case it is the object connected to center port 1 (the Dispatcher) that assigns the operator to the task, so the default settings will work here.
- Click **OK** to close the Properties window.



Step 5: Save the Model, and Test Run

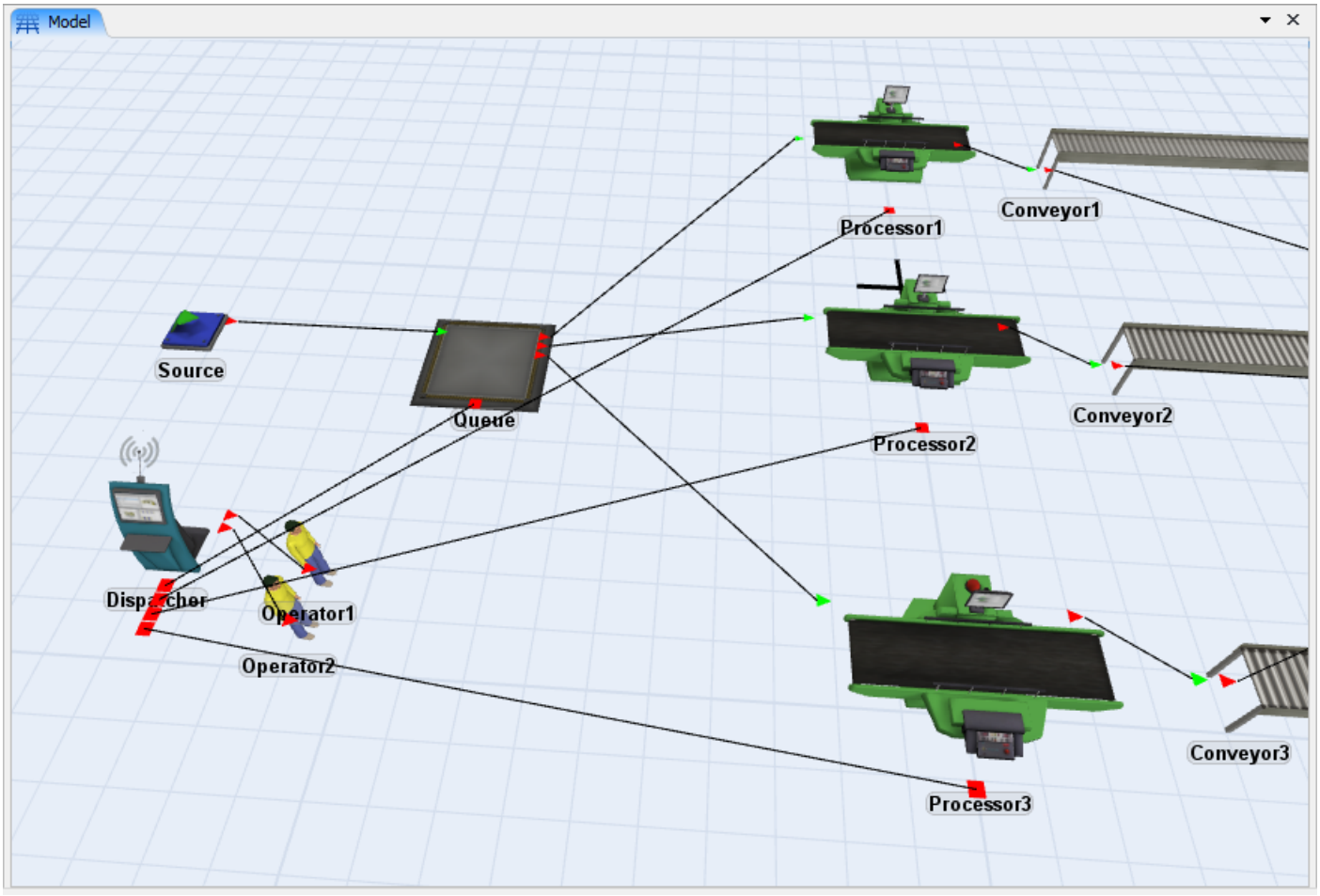
Now we should run the model to make sure that the changes we have made are working.

- **Reset** the model and then **Save** the model by clicking the button on the toolbar.
- **Run** the model to verify that the operators are moving the flowitems from the queue to the testers.

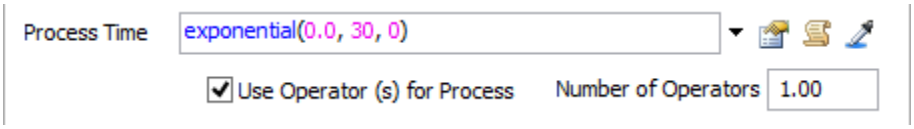
Step 6: Using Operators for the Process Time

In order for the testers to use the operators during processing, a connection must be made between the central ports of each tester to central ports of the dispatcher. Then the process must be changed to require an operator.

- Connect *Dispatcher* to *Processor1, Processor2,* and *Processor3* with center connections (S key).



- Double-click *Processor1* to open its **Properties** window.
- On the **Processor** tab, check **Use Operator(s) for Process.** Number of Operators and Pick Operator will become available.

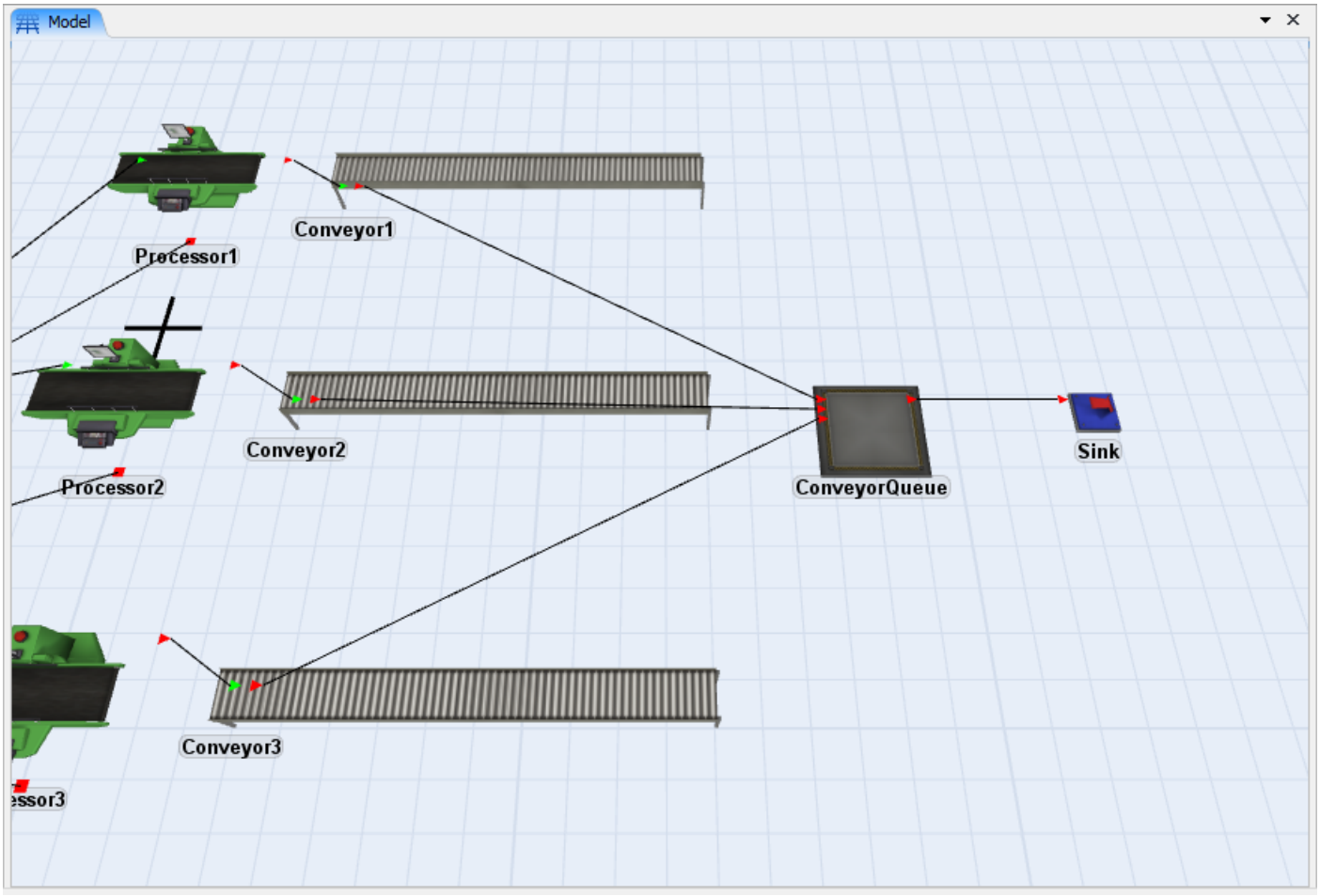


- Click **OK** to close the Properties window.
- Repeat this step for *Processor2* and *Processor3*.

Step 7: Disconnect the Conveyors from the Sink

Before adding the conveyor queue it is necessary to disconnect the input and output port connections between the conveyors and the sink. This is done by pressing and holding the "Q" key on the keyboard and then click-and-dragging from the conveyor to the sink.

- Disconnect *Conveyor1, Conveyor2,* and *Conveyor3* from *Sink* (Q key).
- Create a Queue, place it to the right of the Conveyors, and name it *ConveyorQueue*.
- Connect *Conveyor1, Conveyor2,* and *Conveyor3* to *ConveyorQueue* (A key).
- Connect *ConveyorQueue* to *Sink* (A key).

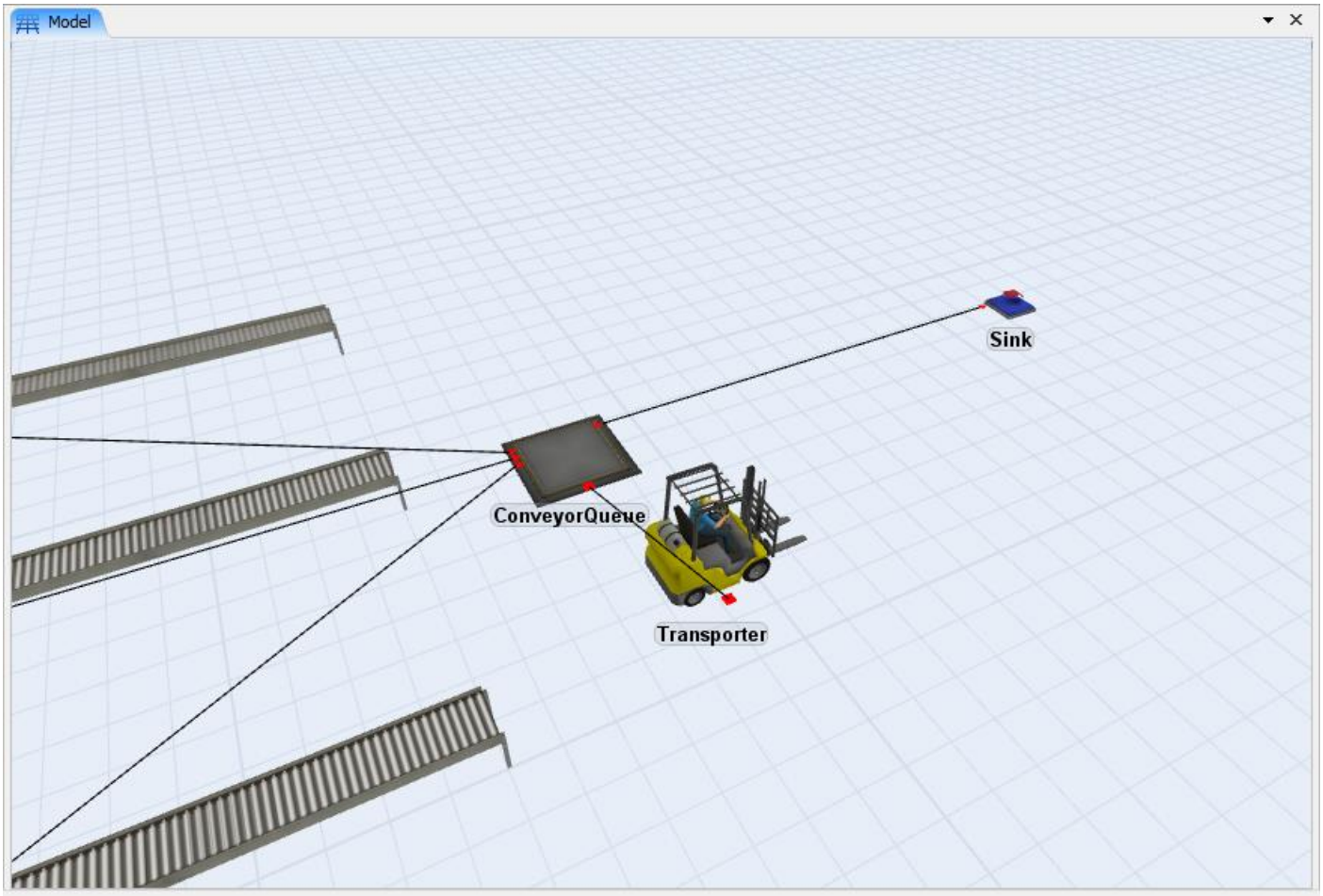


Now that the layout has been revised and the port connections created, the fork truck can be added.

Step 8: Adding the Transporter

Adding the Transporter to the model to move flowitems from the *ConveyorQueue* to the sink is exactly the same as adding operators to move flowitems from the input queue to the testers. There will be only 1 Transporter in the model, so there is no need for a dispatcher. The Transporter will be directly connected to a central port of the *ConveyorQueue*.

- Move *Sink* to the right about 10 units to simulate travel distance.
- Create a Transporter, place it near *ConveyorQueue*, and name it *Transporter*.
- Connect *ConveyorQueue* to *Transporter* with a center connection (S key).

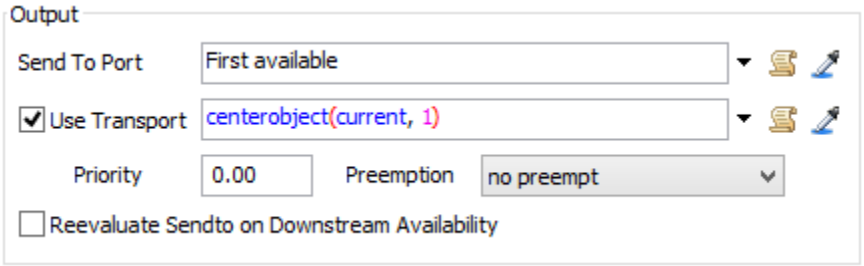


Step 9: Adjust the Queue's Flow Parameters to Use the Transporter

You may define the Queue's flow and transport options by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window.

- Double-click *ConveyorQueue* to open its **Properties** window.
- Click the **Flow** tab and check **Use Transport.** The central port 1 of the queue is already connected so there is no need for any adjustments.

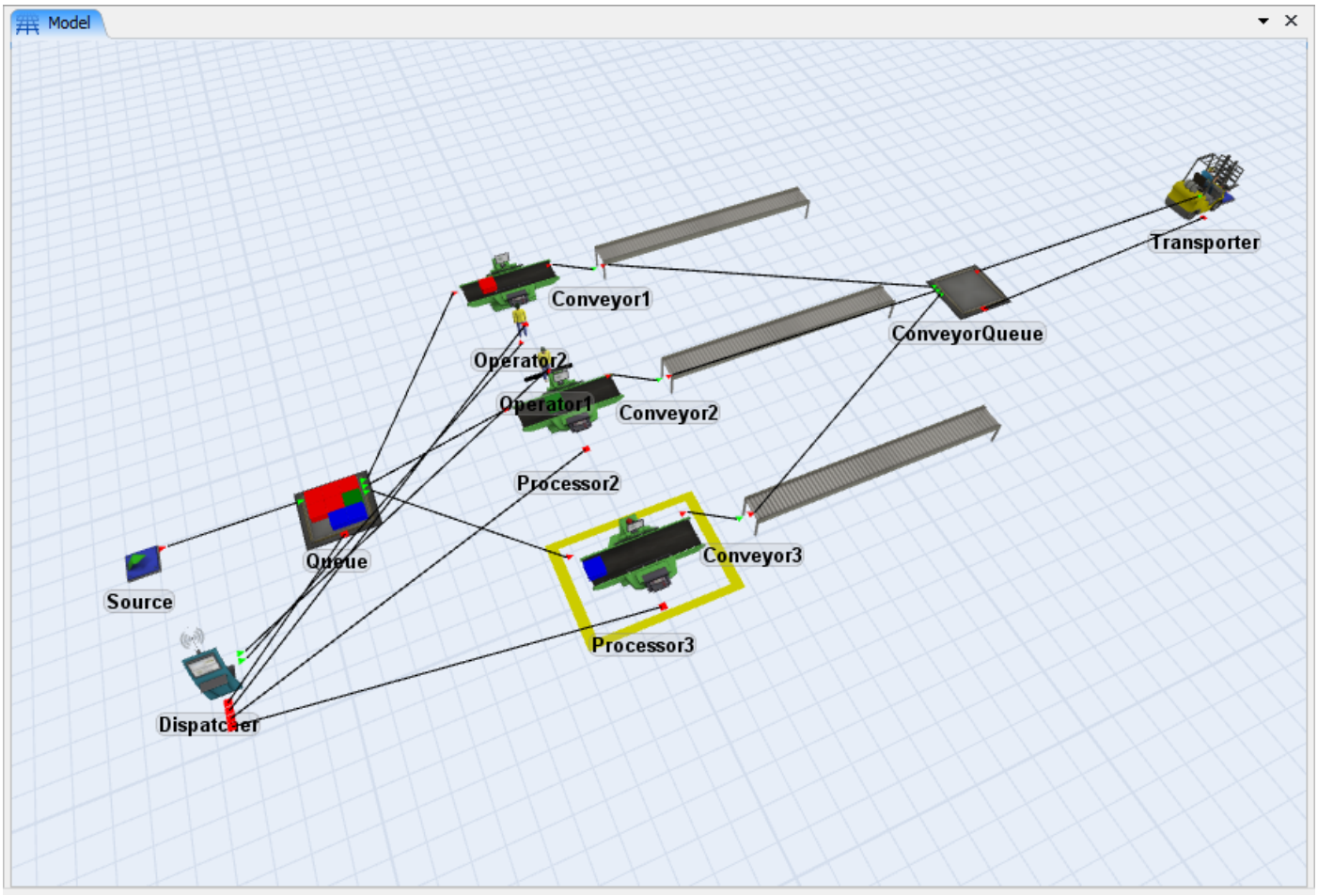


- Click **OK** to close the Properties window.
- **Reset** and **Save** the model.

Step 10: Run the Model

- **Run** the model.

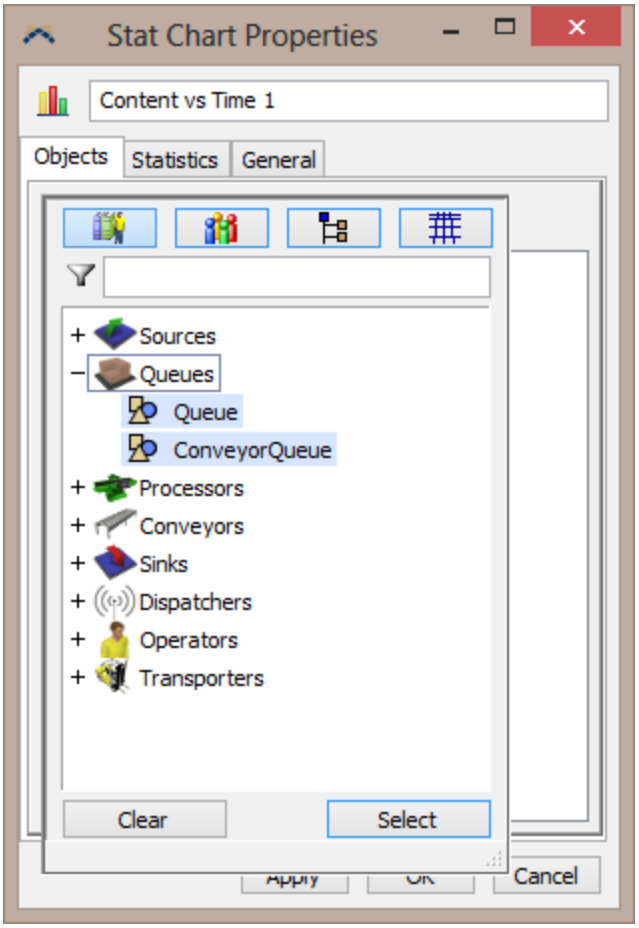
This is the rewarding part of building the model. It's time to check the model to see if it is working the way you want. While the model is running, you can use the animation to visually inspect the model to see if everything is working properly.



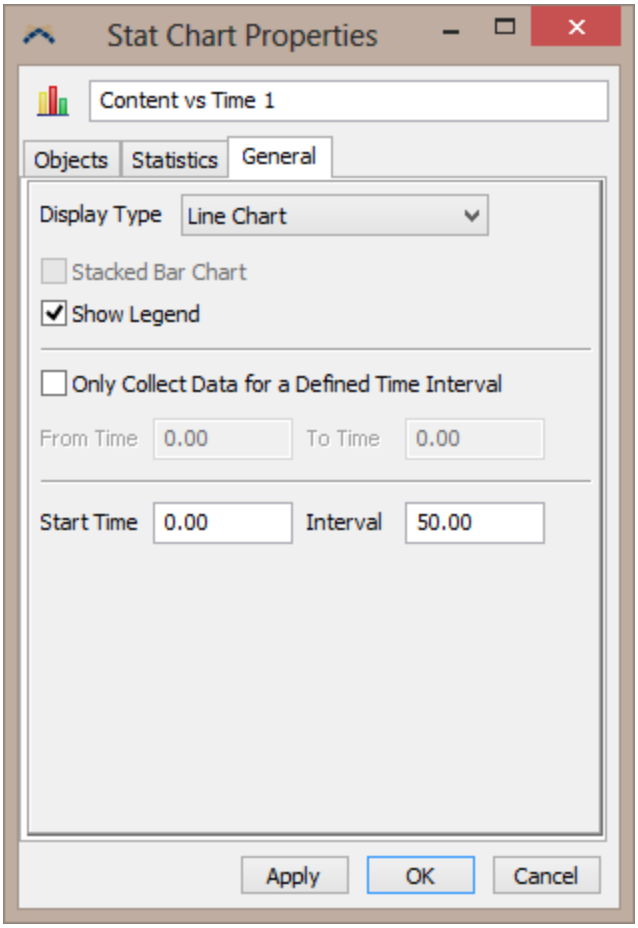
You should see the operators moving back and forth between the queue and the processors. The transporter should be moving flowitems from the queue to the sink. You will notice that when a processor is waiting for an operator to perform the setup, a yellow square is shown under the processor.

Step 11: Analyzing the Output

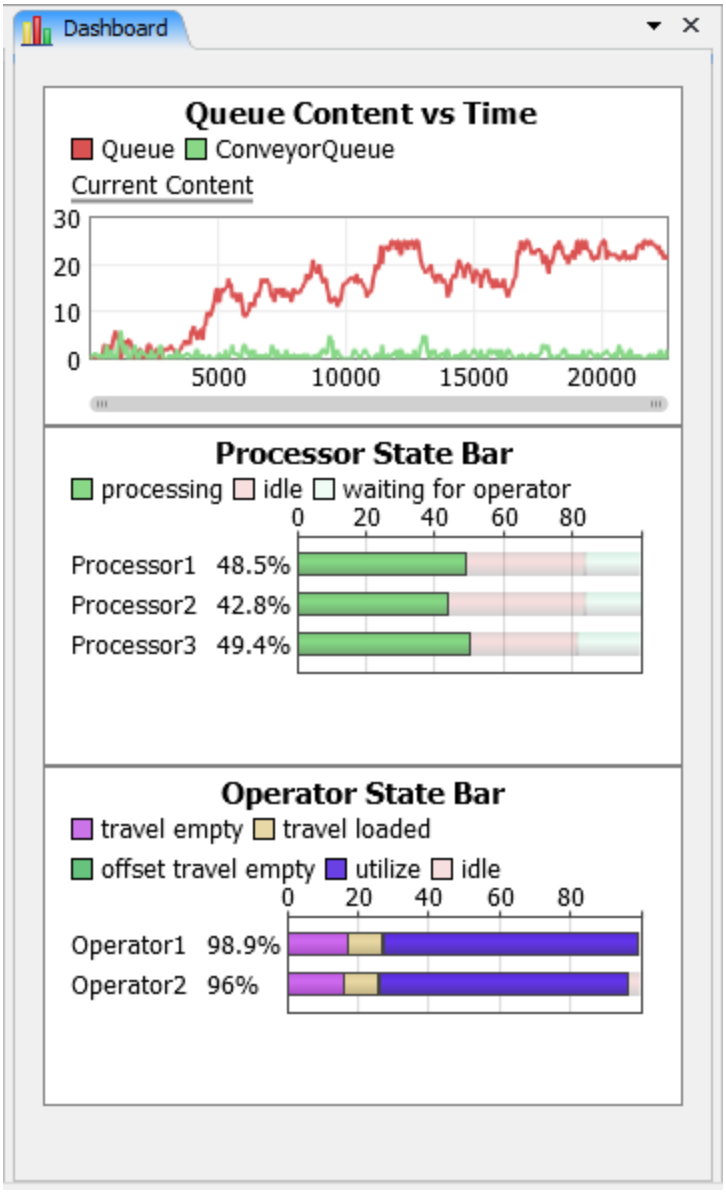
- Select **Dashboard > Add** from the **Statistics** menu. The dashboard window will appear.
- Add a line graph by clicking the button. Select **Content vs Time**. The object selection window will appear.
- Click the to add objects to the graph. Add *Queue* and *ConveyorQueue*. Click **Select** .
- Change the chart name to **Queue Content vs Time** and click **OK**. A blank chart should appear in the dashboard.



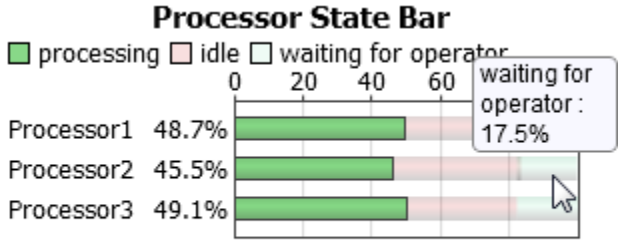
- Add a legend by double-clicking the chart in the dashboard. Select the **General** tab and check the **Show Legend** checkbox. Click **OK** to return to the dashboard.



- Now add a bar chart by clicking the button and selecting the **State Bar** option.
- Add all of the processors to the chart.
- Change the chart name to **Processor State Bar** and click **OK**. Another blank chart should appear in the dashboard.
- Add another state bar chart to the dashboard for the operators. Follow the same steps as above, but add all the operators to the chart, instead of processors. Change the chart name to **Operator State Bar** and click **OK**.
- **Reset** and **Run** the model. The graphs will dynamically update.



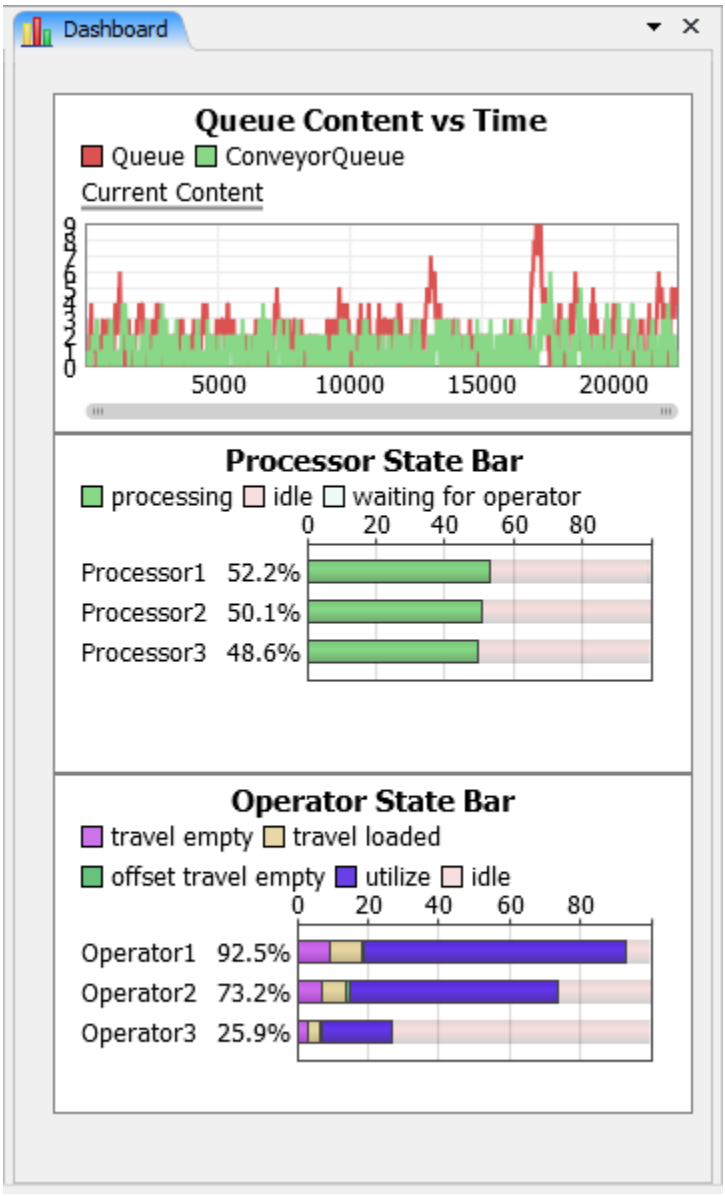
- Hold the mouse over one of the bars in the **Processor State Bar** graph. A ToolTip will appear and give more information.



It becomes obvious that if you add one more Operator the model will run better. Even though the flowitems may still back up at the input queue it will be in its optimum configuration with the addition of a third operator.

- Create an Operator, place near the other two, and name it *Operator3*.
- Connect *Dispatcher* to *Operator3* (A key).
- Double-click the *Operator State Bar* chart and add *Operator3*.

- **Reset**, **Save**, and **Run** the model.



With one more operator, the processor wait time goes down and that the *Queue* stays a a much lower content level.

This ends Lesson 2. Congratulations! Can you go the extra mile?

To continue the tutorial, go to Lesson 2 Extra Mile.

Lesson 2 Extra Mile

1. Introduction
2. Step-By-Step Model Construction

Introduction

This extra mile session is designed to teach the modeler how to add the extra touch to make the model show data and information as the model is running. In this lesson we will look at how to add charts, graphs, and 3D text to the model you finished in lesson 2.

What You Will Learn

- How to add a content graph for the Queue
- How to add a histogram to show the wait time for the Queue
- How to add a pie chart to show the state profile for each operator
- How to add 3D visual text to show the average wait at the Conveyor Queue
- How to position the graphs, charts, and text for best viewing

New Objects

In this lesson you will be introduced to the VisualTool and Dashboard objects.

Approximate Time to Complete this Lesson

This lesson should take about 20-30 minutes to complete.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

Building Model 2 Extra Mile

To start building model 2 extra mile you will need to load model 2 which you saved from the last lesson.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Load Model 2

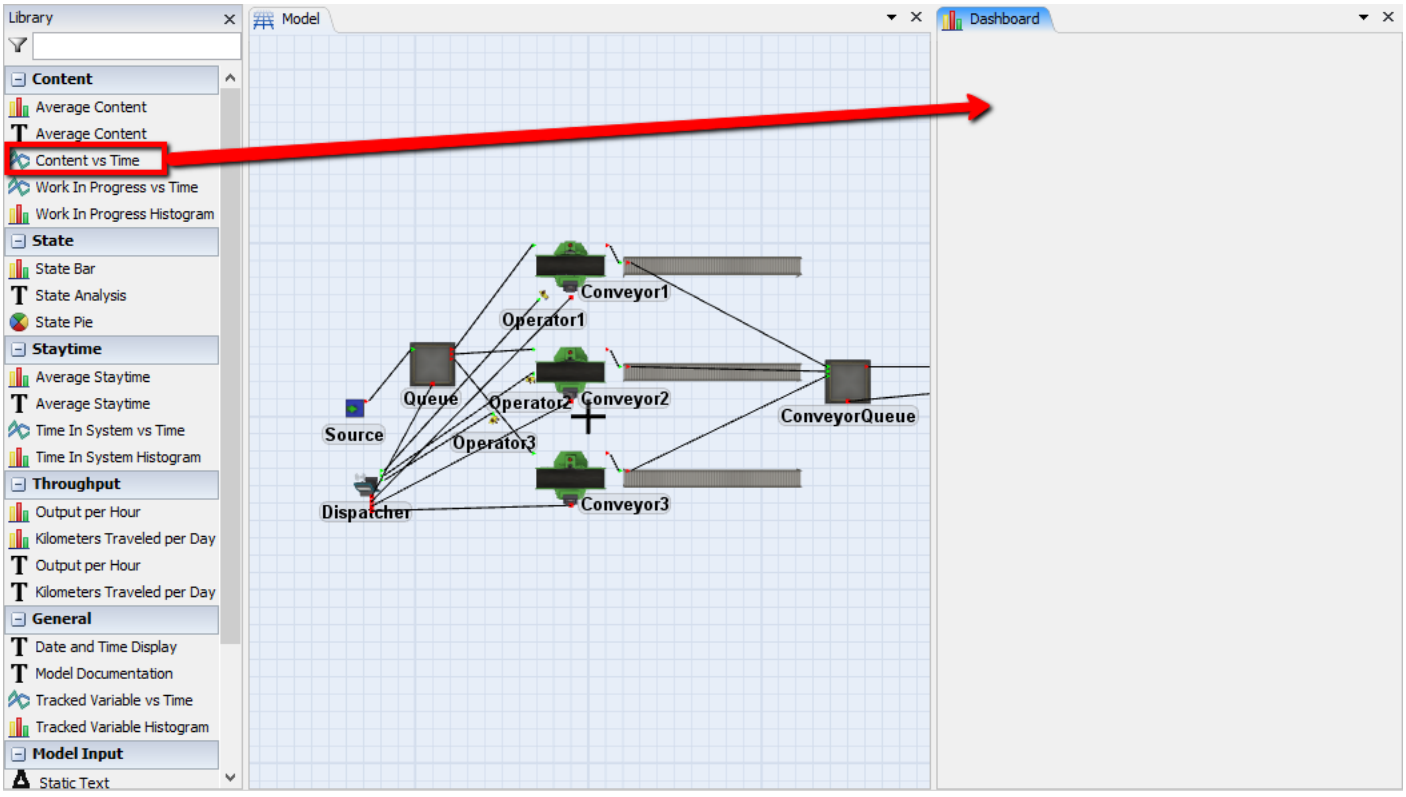
- Load Model 2 if it is not already open.

Step 2: Save Model as "Model 2 Extra Mile"

- Go to the menu option **File** > **Save Model As**... to save your model under a new name.

Step 3: Add a Dashboard to Show the Content of the Queue

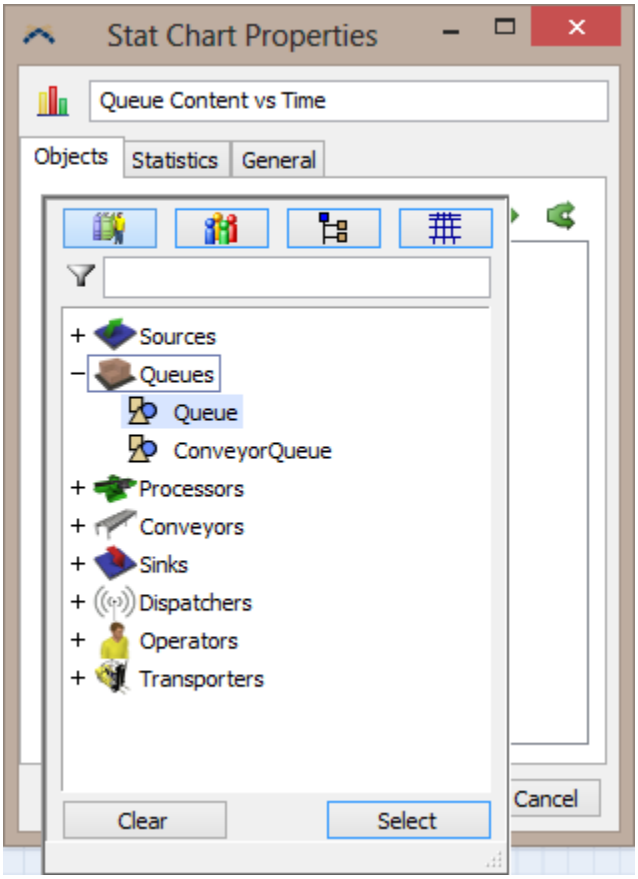
- Create a Dashboard, by clicking Statistics->Dashboards->Add, and then dragging the "Content vs Time" object from the library into the dashboard pane.



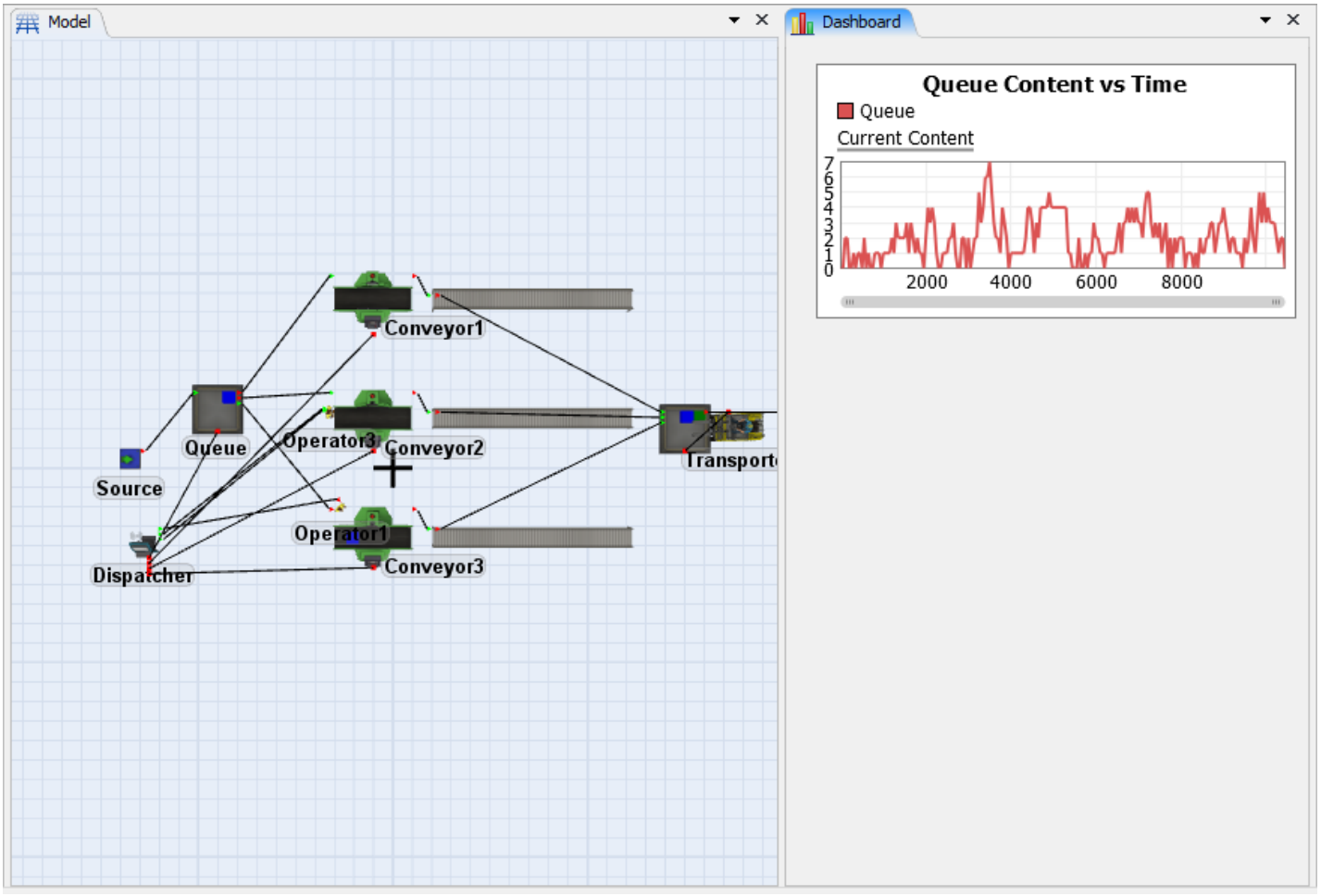
Step 4: Adjust the Parameters of the Dashboard to show a Content Graph of the Queue

- In the **Dashboard** properties, Push the **Green Plus** on the **Objects** tab. You will now be able to select the Queue to show its Content vs Time.
- Push the plus button next to **Queues**, select **Queue**. Push the **Select** button below to complete your selection.

- Change the name of the Dashboard to **Queue Content vs Time**
- Click **OK** to apply the changes and close the window.

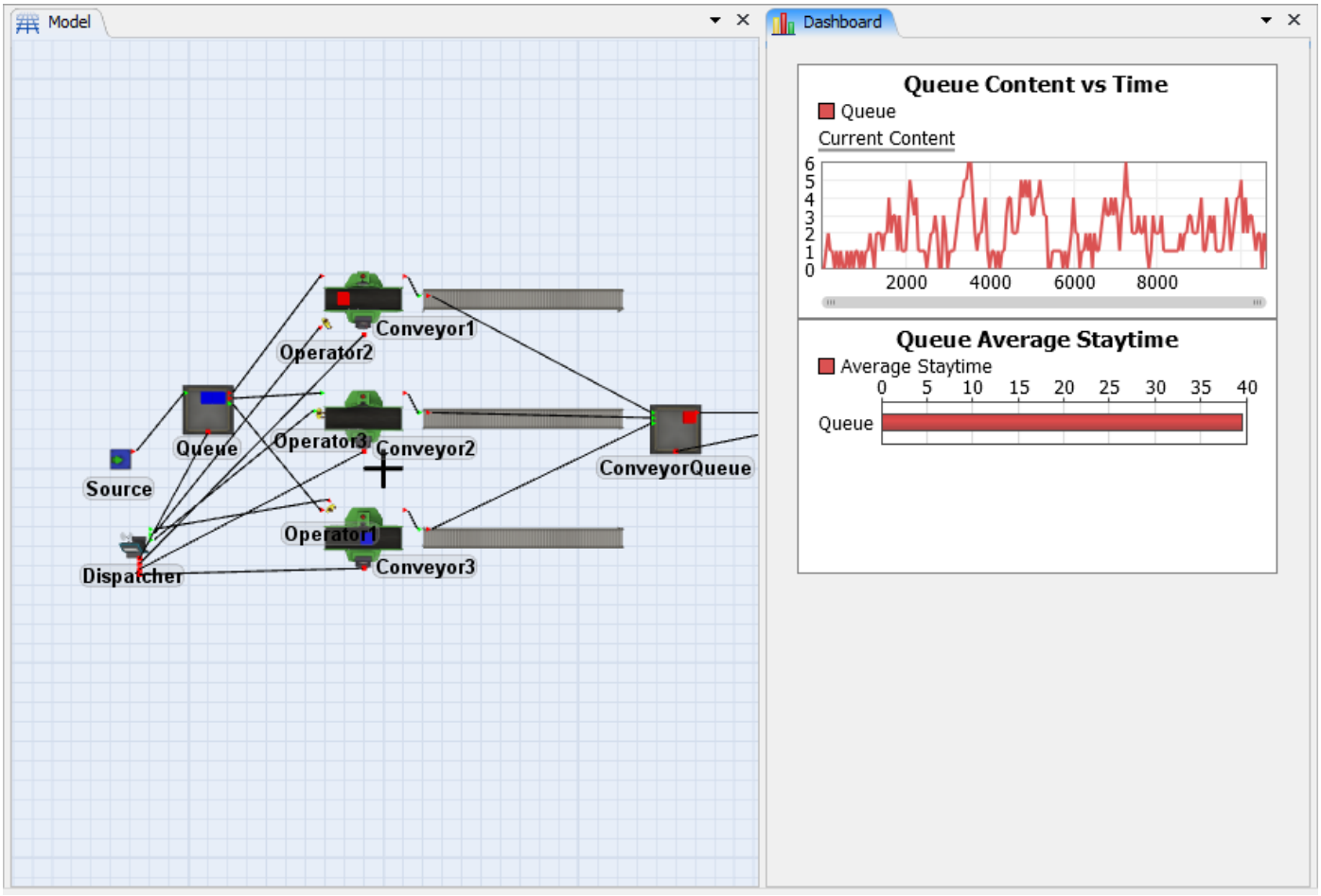


Alternatively, you can click on the and then click on the Queue in the 3D view to add it to your dashboard.



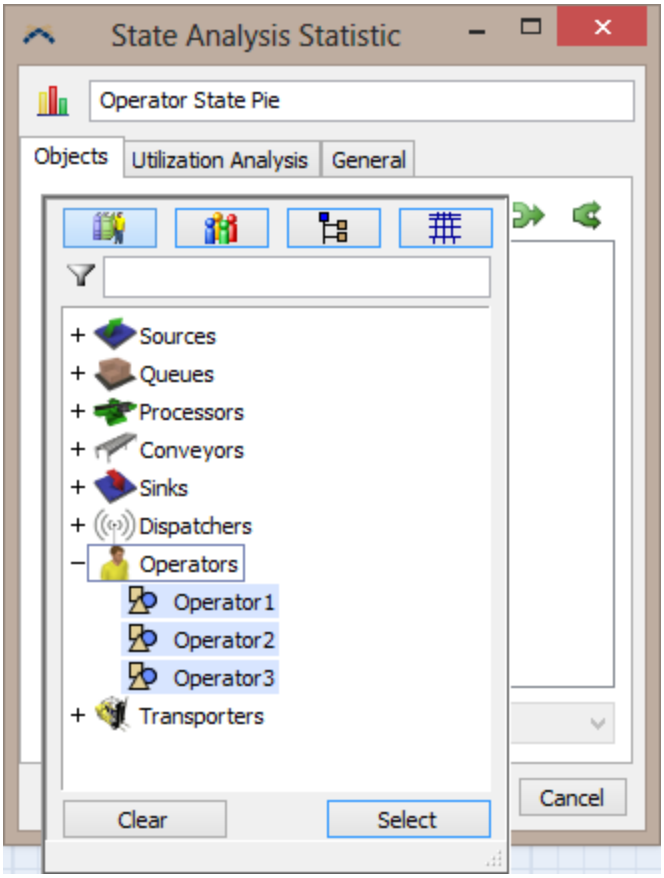
Step 6: Add a Dashboard to show the Average Staytime of the Queue

- Following the same steps from steps 4, add a new **Dashboard** and place it below the content graph. The only difference will be that you will drag an **Average Staytime Bar Graph** instead of "Content vs Time" from the **Library** and the **Graph Title** will be **Queue Average Staytime**.

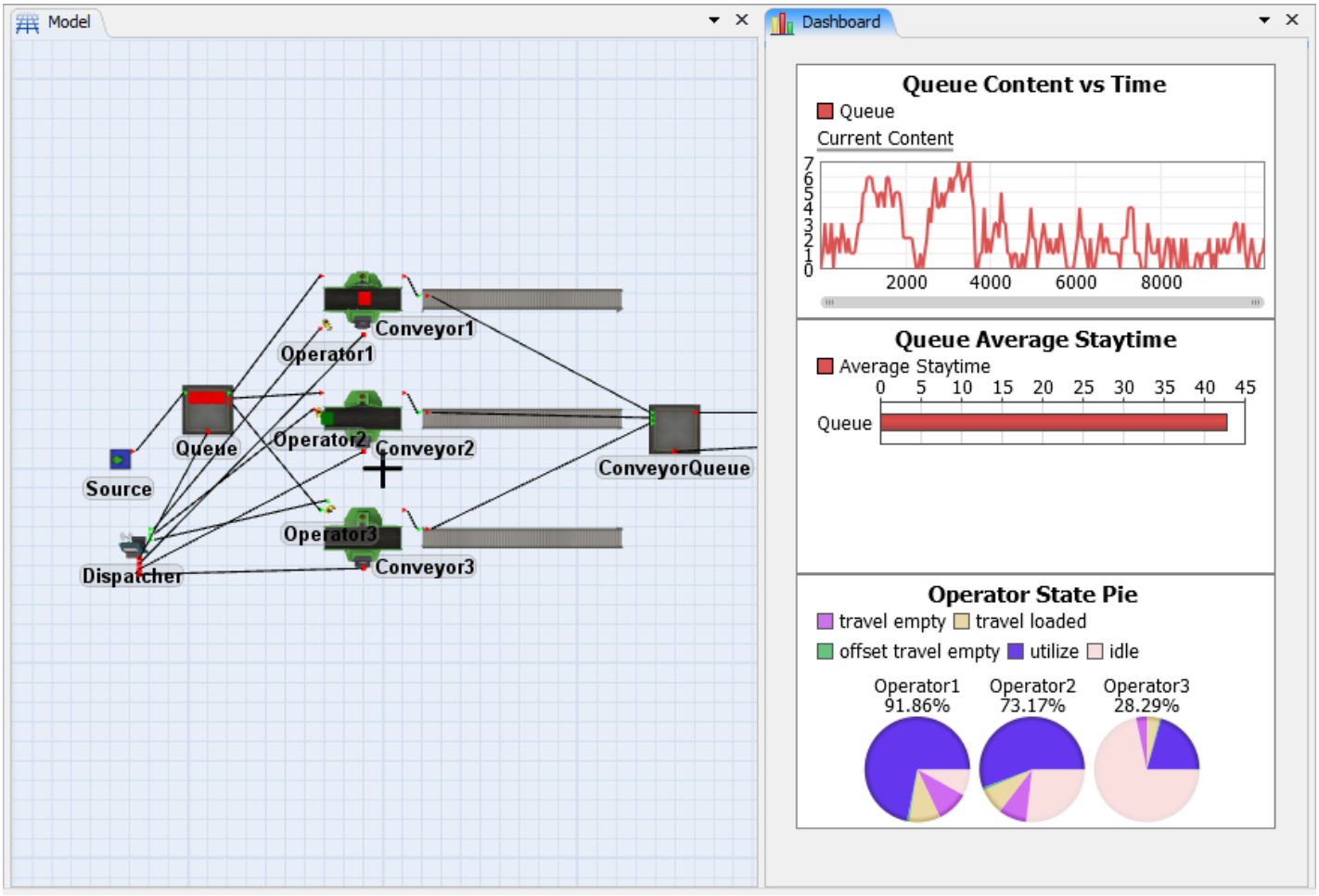


Step 7: Add a State Pie Chart for each Operator

- Following the same steps from step 4, add a new **Dashboard** for each Operator. Select **State Pie**instead of "Content vs Time" from the **Library** and set the **Graph Title** to be **Operator State Pie**.



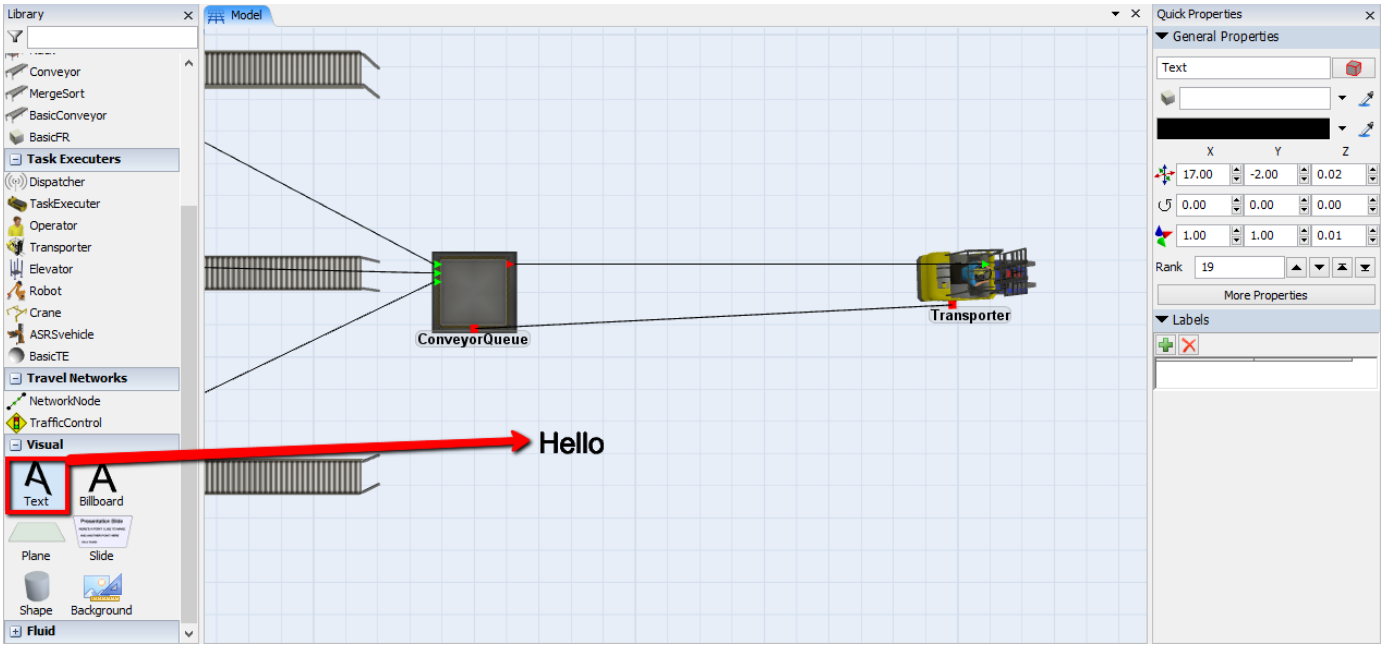
When reset and run, the pie charts should look something like this:



Step 8: Adding 3D Text to the Model

There is another way to add information to the model that can show performance measures. You can place 3D text at strategic points in the layout to show what is happening while the model is running. This is done using a VisualTool. In this model we will add 3D text to show the average wait time of flowitems in the "Conveyor Queue".

- Add a **VisualTool** to the model, place it by **ConveyorQueue**, and name it *Text*.

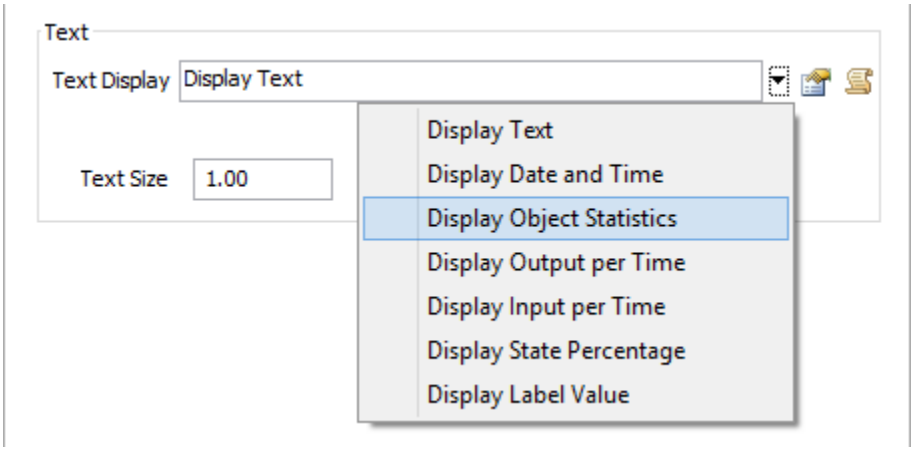


- Double-click the **VisualTool** to bring up its **Properties** window (see Figure 2-45).
- In the Text Display list select the **Display Object Statistics** option. The code template window will appear. Change the parameters text to be as follows:

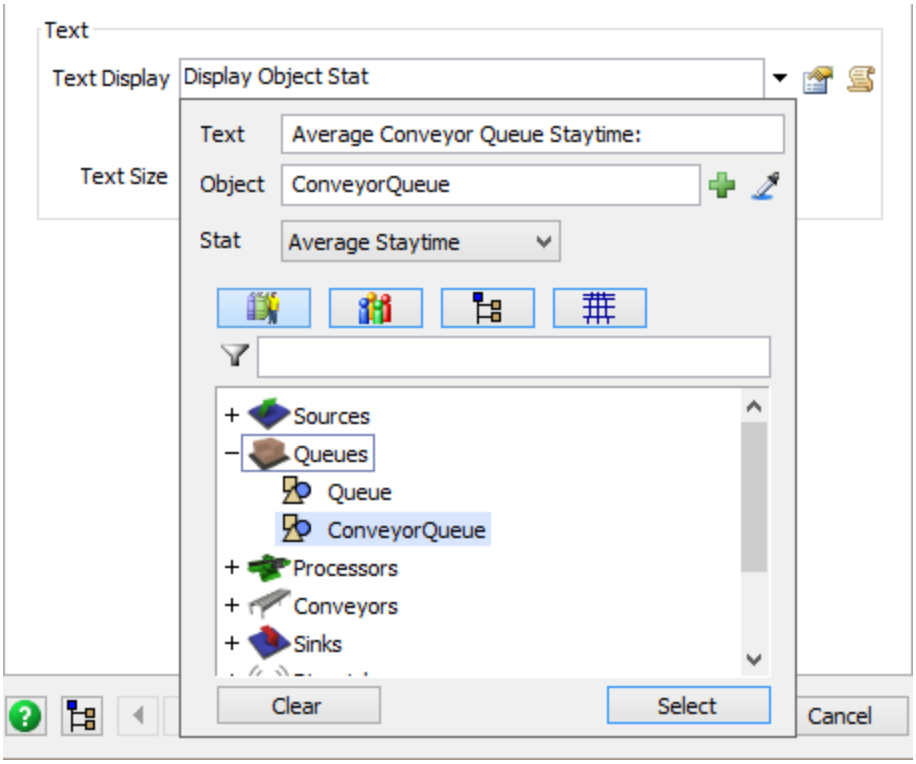
Text: "Average Conveyor Queue Staytime: "

Object: centerobject(currrent,1)

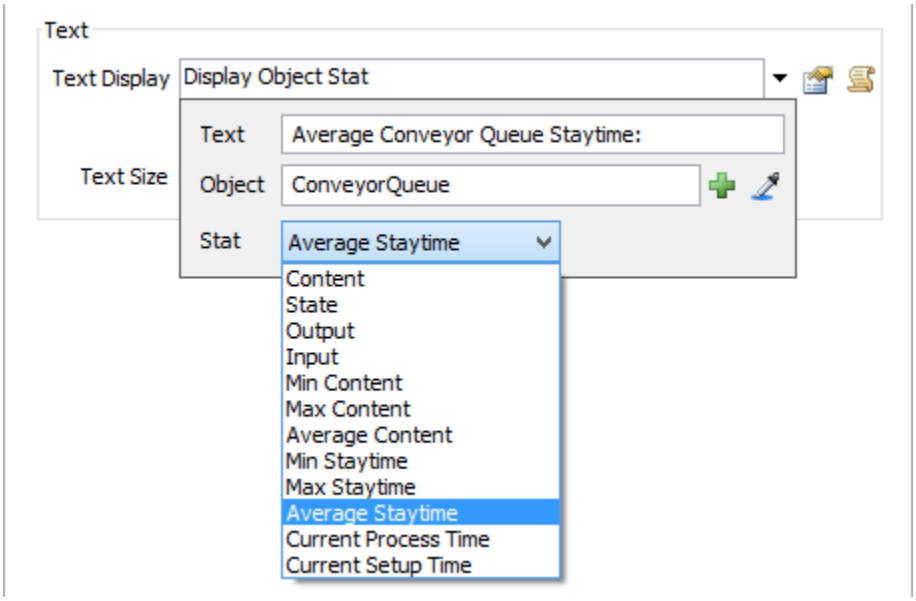
Stat: Average Staytime



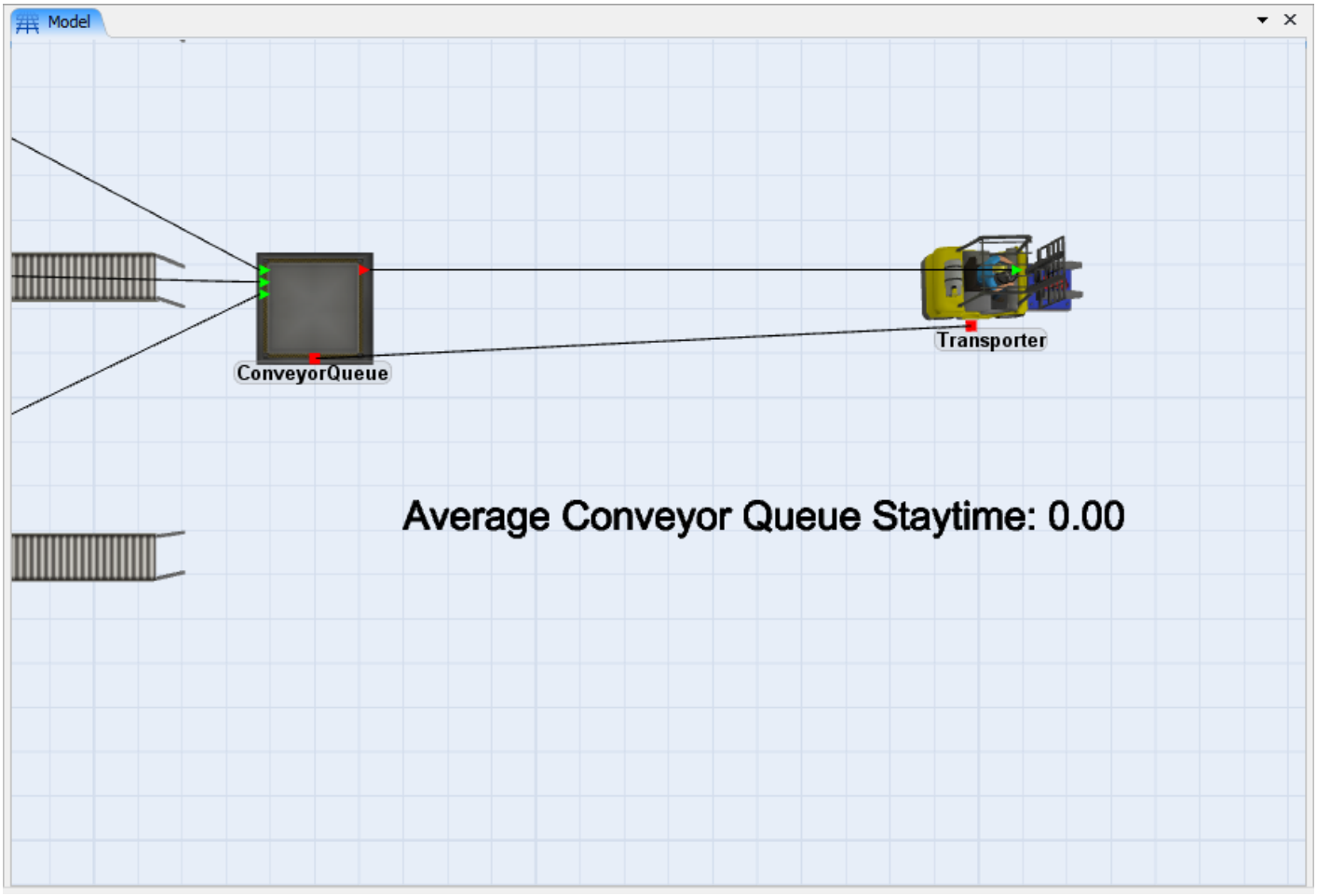
- For the **Object:** value, click the Green Plus sign, then click on the plus sign next to **Queues**, and select **Average Staytime**. Then press **Select**.



- For the **Stat:** value, click the Dropdown Menu, and select **Average Staytime**.

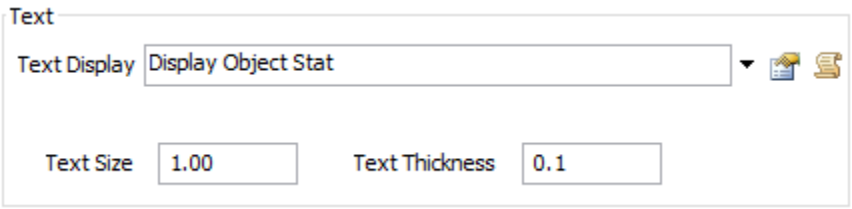


NOTE: To click on the VisualTool, click directly on the 3D text that is showing.

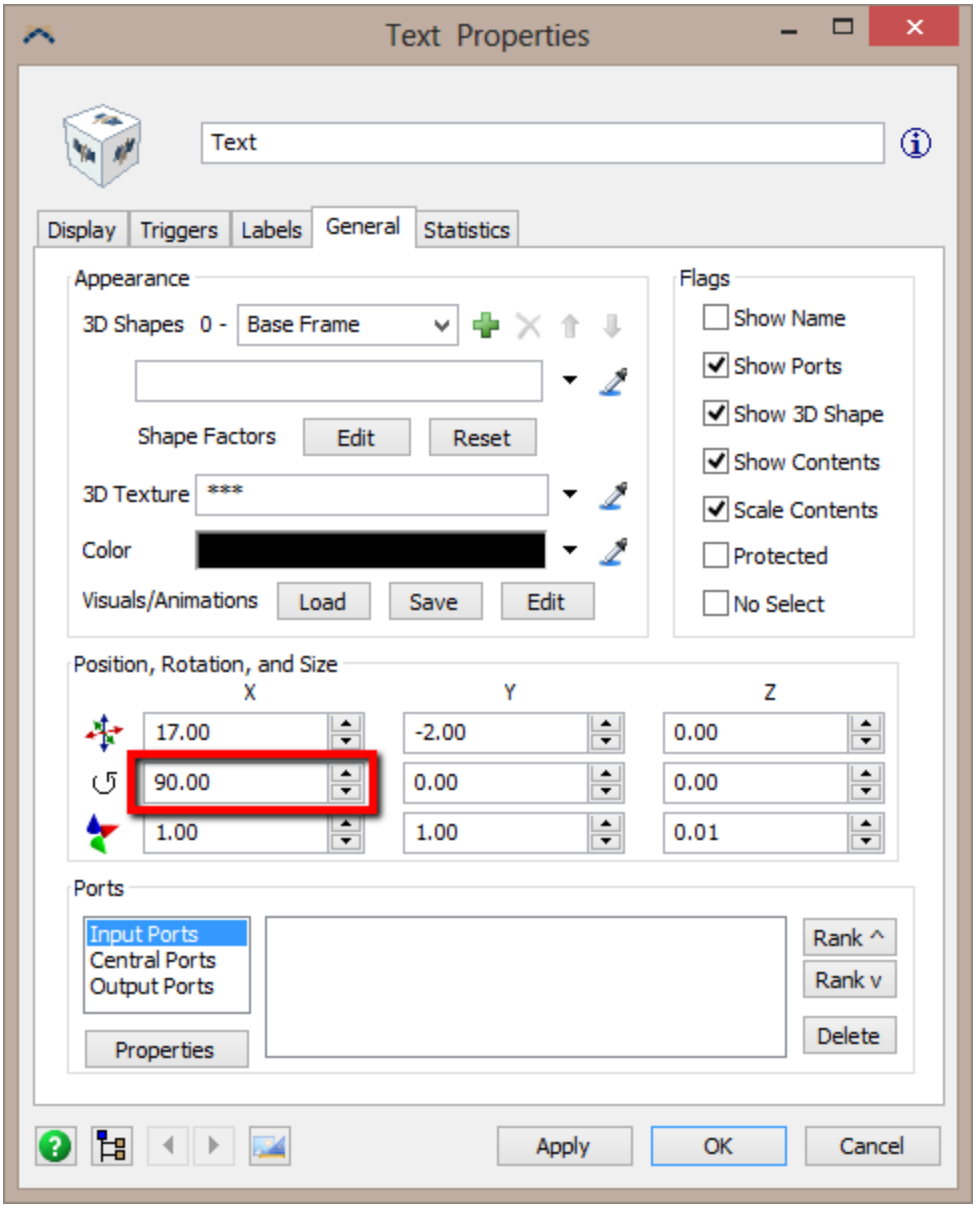


At this point you may want to adjust the display of the text. The text size is set to 1 by default, and you may want to make it smaller. You may also want to have the text hover over the queue. To make the text smaller, type the desired size in the Properites window of the VisualTool. You may also adjust the thickness to give the text a 3D appearance.

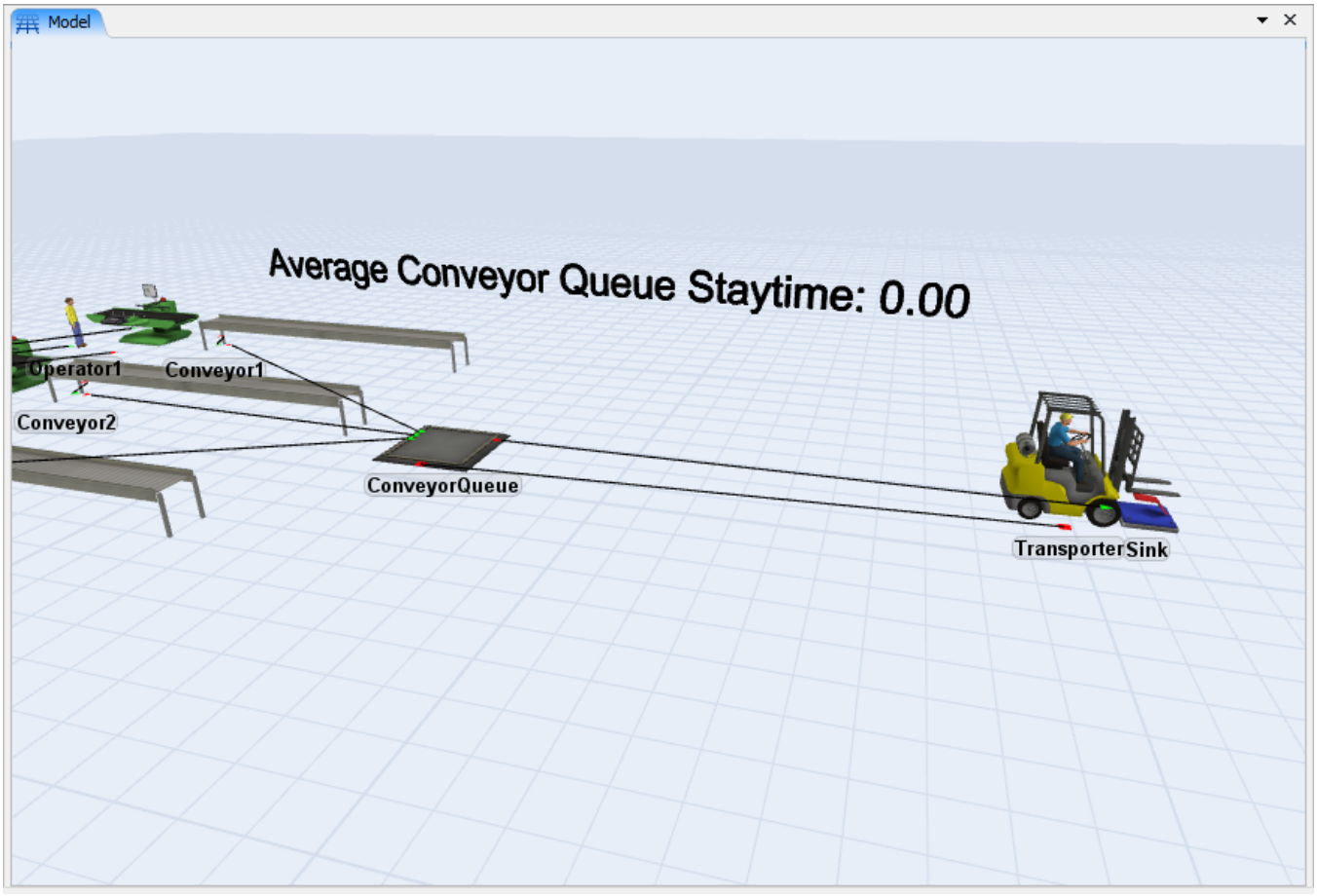
- Double-click the VisualTool to open its **Properties** window. On the **Display** tab, change **Text Thickness** to 0.1.



- Click the **General** tab.
- Change **Rotation X** to 90.
- Click **OK** to close the Properties window and apply the changes.

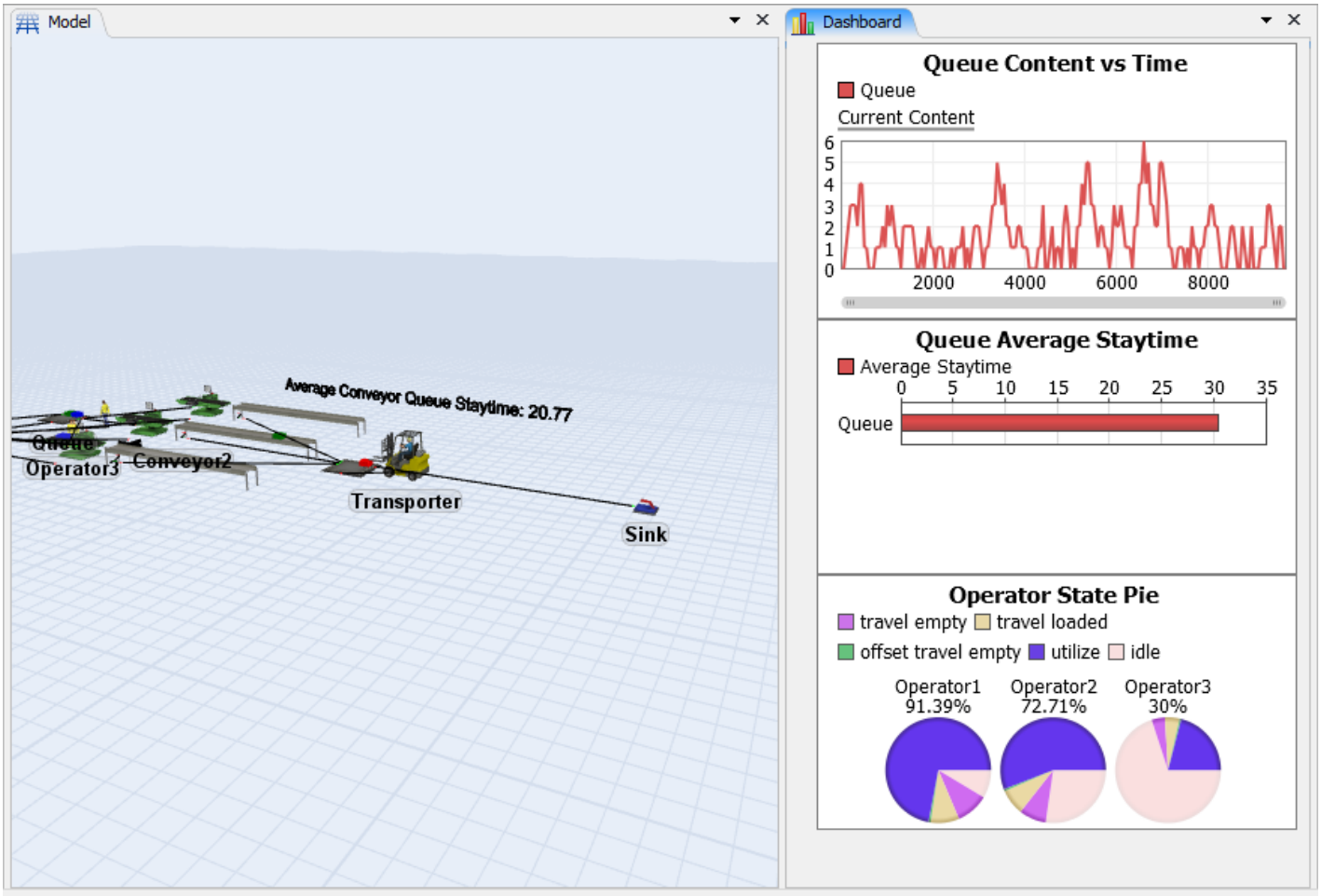


The text will now be rotated in the model. Use you mouse to select and position the text as desired. Remember that the z-position of the text can be controlled by selecting the text with both the left and right mouse buttons and moving the mouse forward and back, or selecting the text and then rolling the mouse wheel to move the text up or down.



Step 9: Reset, Save and Run

- **Reset** and **Save** the model. You are then ready to **Run** the model and look at the graphs, charts, and 3D text you have just added.



This ends the "Model 2 Extra Mile" lesson. As you can see, it is very easy to add powerful 3D reporting visuals to your simulation models.

To continue the tutorial, go to Lesson 3.

Lesson 3 Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

Lesson 3 introduces the **Rack** and **NetworkNode** objects. You will have a chance to work with spline points, conveyors, advanced statistics, and global tables. With lesson 3 you will be introduced to the Experimenter, which allows you to do multiple runs and multiple scenario analyses of your model. Lesson 3 will use the model from lesson 2 as a starting point. Make sure you have completed lesson 1, and lesson 2 before starting lesson 3.

Lesson 3 assumes you have worked through lessons 1 and 2 and are familiar with properties windows. In the previous lessons, almost every step was illustrated to make sure you had a complete understanding of the steps needed to build the model. In lesson 3 some of the simple tasks such as adding a new object to the model and entering basic properties will still be identified in the step-by-step description, but screen shots may not be provided.

Note: If you are using the Evaluation version of FlexSim, you will not be able to complete this model. This lesson exceeds the number of allowed objects in the Evaluation version.

What You Will Learn

- How to use global tables to define routings
- How to set up a travel path network for a transporter
- How to create splines in a travel path network
- How to create a custom output report
- How to execute multiple runs of the model

New Objects

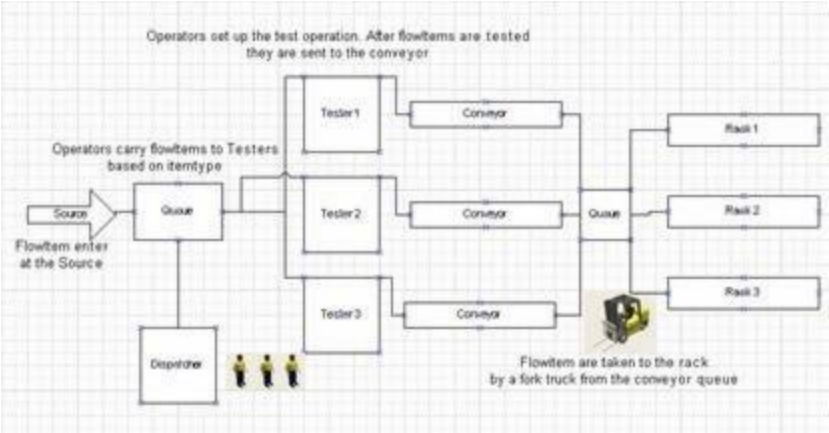
In this lesson you will be introduced to the Rack and NetworkNode objects as well as Spline Points.

Approximate Time to Complete this Lesson

This lesson should take about 45-60 minutes to complete

Model 3 Overview

In model 3 the sink will be replaced with 3 racks that will be used to store the completed flowitems prior to shipping. You will change the physical layout of conveyors 1 and 3 to bend at their ends to that flowitems are conveyed closer to the queue. Using a global table for reference, all itemtype 1 flowitems will be sent to rack 2, all itemtype 2 flowitems will be sent to rack 3, and all itemtype 3s flowitems will be sent to rack 1. Using the NetworkNode object, you will set up a path network for the fork truck to use as it transports flowitems from the conveyor queue to the racks. You will also set up a multiple run simulation using the Experimenter to show statistical variance and calculate a confidence interval for key performance measures.



Model 3 Data

Modify conveyor 1 and 3 to convey flowitems closer to the conveyor queue.

Routing from conveyor queue to racks: Use a global table to specify the routing for flowitems as follows:

- Itemtype 1 to rack 2
- Itemtype 2 to rack 3
- Itemtype 3 to rack 1

Set up a path network for the fork truck to travel on between the conveyor queue and the racks.

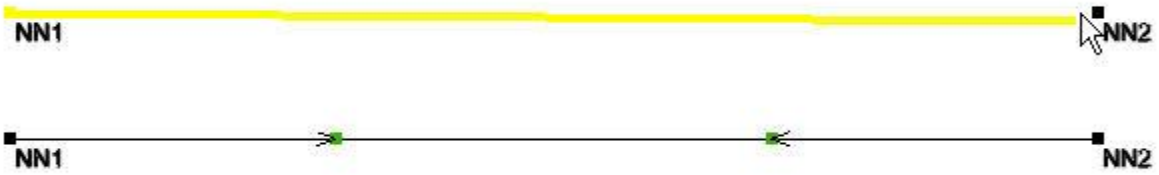
Set up a flypath for a fly-through model presentation.

New Concepts

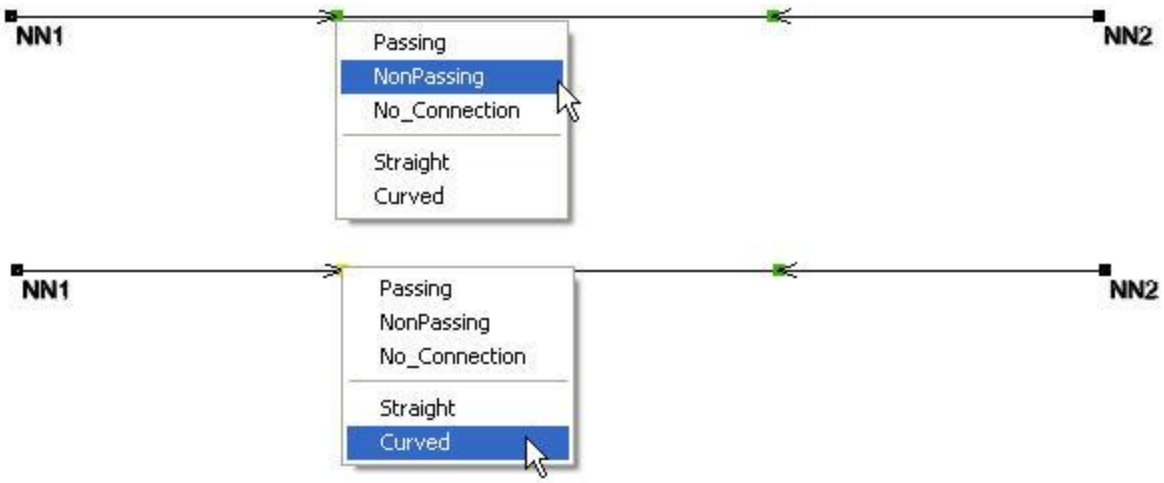
Spline Control Points

Spline control points are used in FlexSim when laying out a travel path network. FlexSim uses spline technology to give you a convenient method to add curves, inclines, and declines to NetworkNode paths.

When two NetworkNodes are placed in the model view and connected together by click-and-dragging with the "A" key, a black line will be displayed. There are also two green boxes with arrows at about 1/3 and 2/3 of the way down the line.

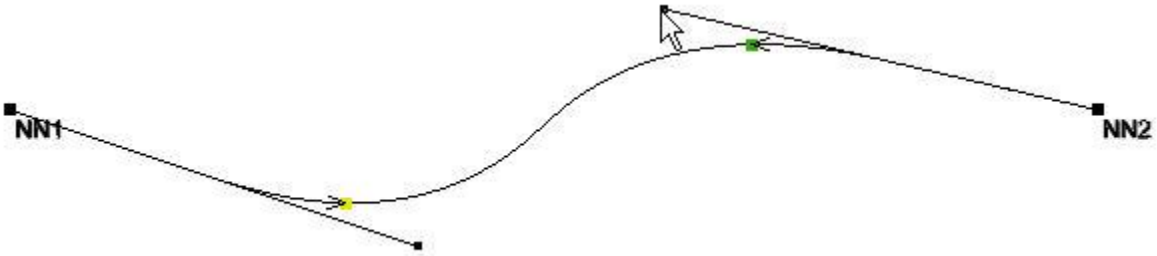


These green boxes indicate the attributes of the path going in the indicated direction. Green means it is a passing path, yellow means it is a nonpassing path, and red means it is a "no connection" or in other words it is a one-way path going the other way. To switch between these colors, you can right click on the node and select an option. You can also make the path a curved path by selecting the "Curved" option in the drop down menu. This will create two spline control points that you can move to create a curved path. You can also configure how connections are made by default using the Travel Networks Menu.

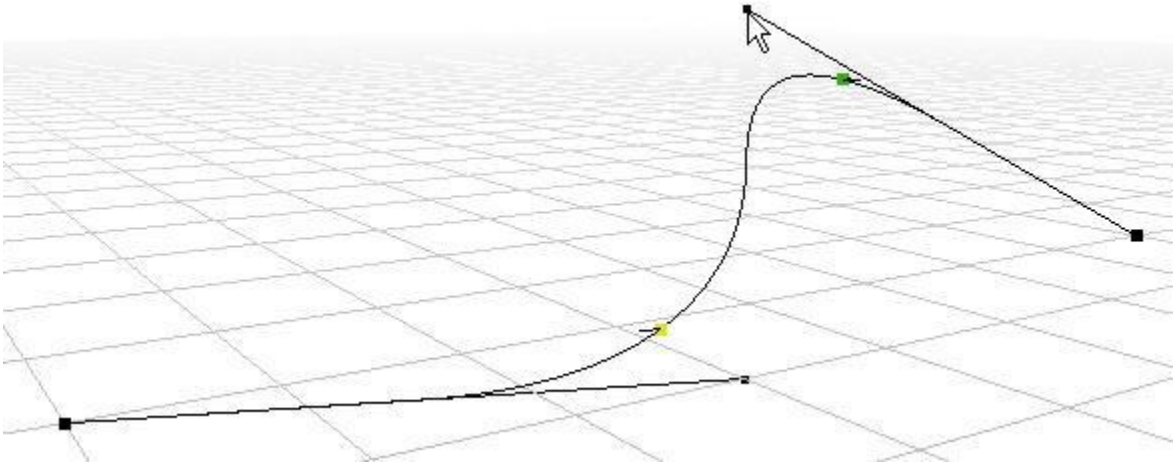


Spline Control Point parameters

Once you have created a curved path, move the small control points with the mouse.



To change the Z height of the spline control point, click on it and roll the mouse wheel up and down.



NetworkNodes can be configured to specify the direction of the path. Again, you can use the right-click menu on the colored box, or, for a quicker method, you can hold down the "X" key and click on the colored box.

When a path has been configured using spline paths, the travelers that use the path will automatically follow the spline that has been defined. The display of the spline control points, as well as the colored

boxes, can be toggled on and off by holding down the "X" key and clicking on one of the NetworkNodes in the path network. Multiple "X" clicks will toggle between several different visual modes for the network.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

Building Model 3

To start building model 3, you will need to load model 2 from the last lesson.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Load Model 2

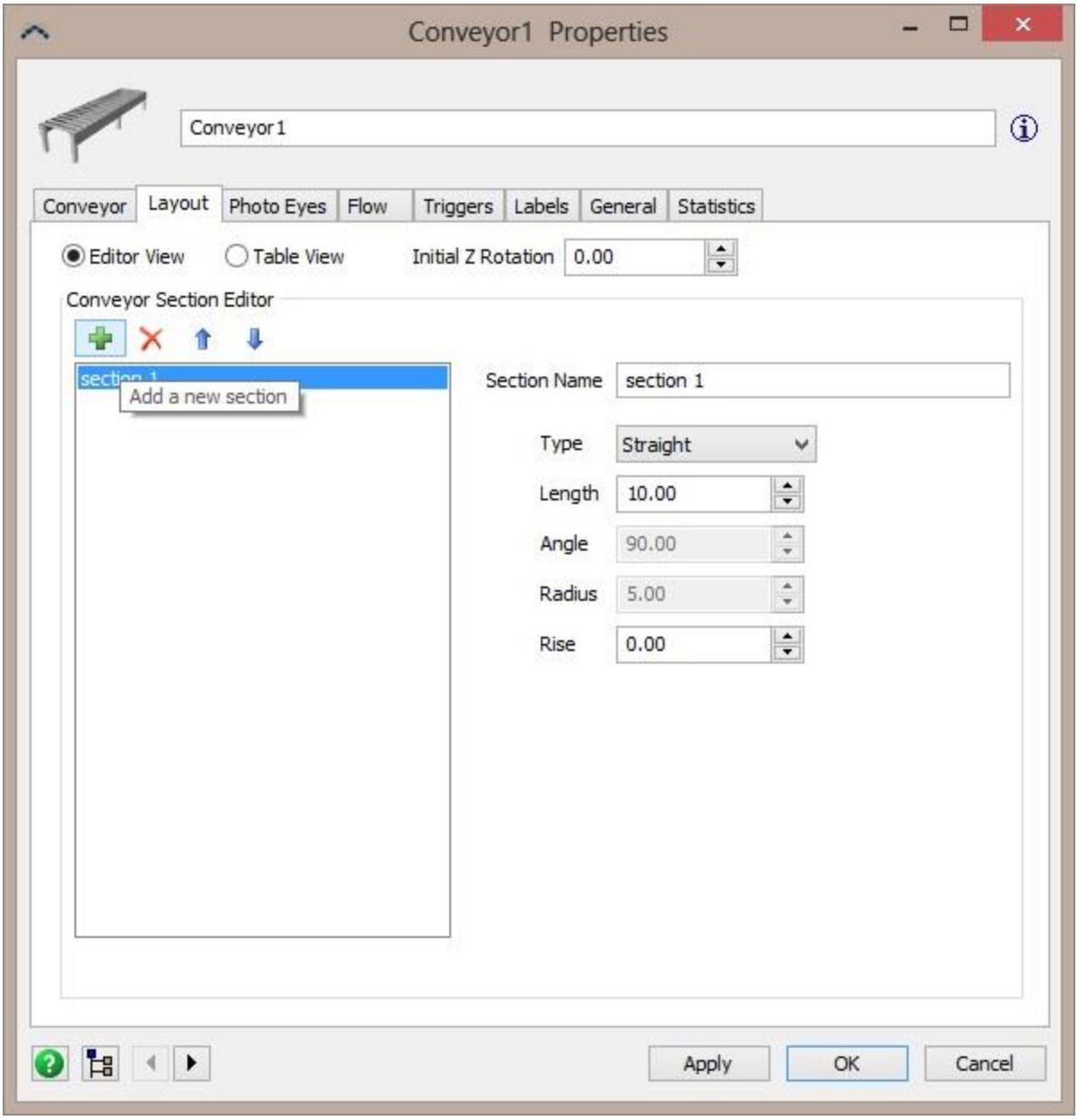
- Load model 2 if it is not already open.

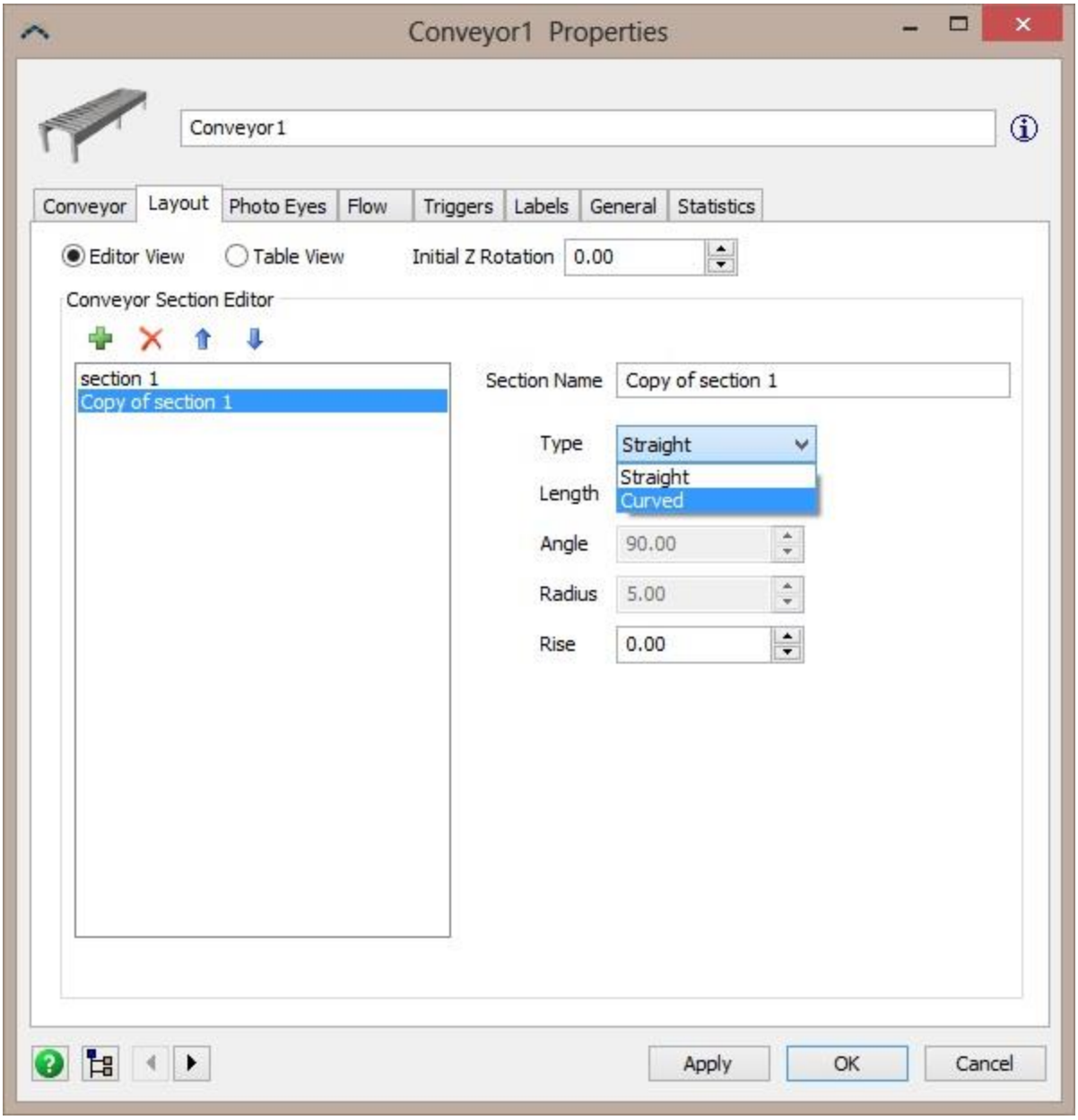
Step 2: Reconfigure the layout of Conveyor1 and Conveyor3

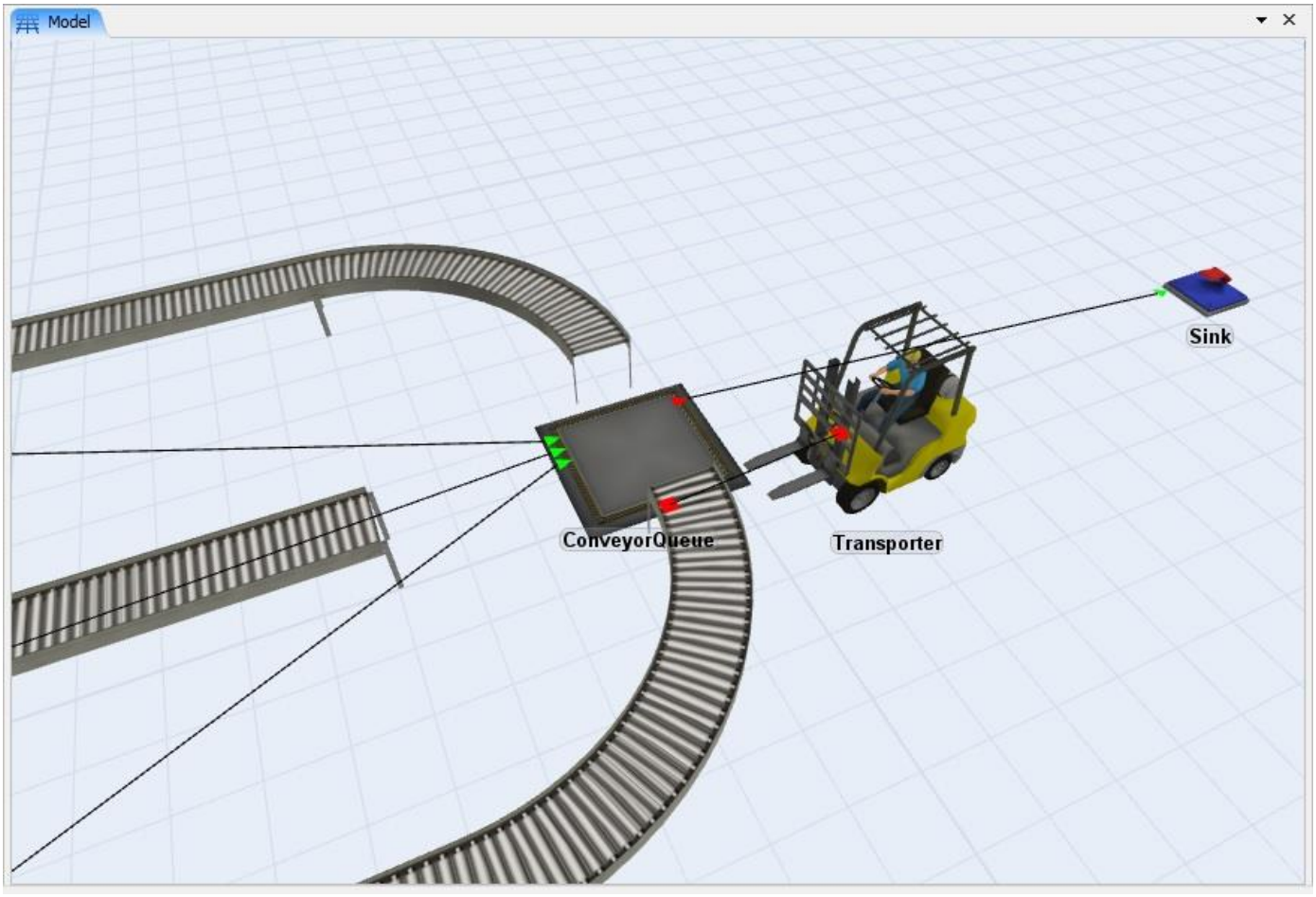
Now we are going to change the layout of the three conveyors so that they have a curved section at the end to convey the flowitems closer to **ConveyorQueue**. You might want to experiment with this Layout tab to create complex curves and rises. Have some fun!

- Double-click *Conveyor1* to open its **Properties** window.
- Click the **Layout** tab. You can find more information about this page here.
- Click **Add** to add a section of conveyor at the end.
- Select **Curved** from the **Type** list.
- Edit the angle and radius values in the table so that **Conveyor1** curves into the **ConveyorQueue** as shown below.
- Repeat this process for *Conveyor3.*

Note: The section of conveyor that you have selected in the list on the left will appear highlighted in a darker color in the model view, so that you can easily tell which section you are working on.


Conveyor1 Properties
Conveyor1
Conveyor
Layout
Photo Eyes
Flow
Triggers
Labels
General
Statistics
Editor View
Table View
Initial Z Rotation
0.00
Conveyor Section Editor
section 1
Add a new section
Section Name
section 1
Type
Straight
Length
10.00
Angle
90.00
Radius
5.00
Rise
0.00
Apply
OK
Cancel


Conveyor1 Properties
Conveyor1
Conveyor
Layout
Photo Eyes
Flow
Triggers
Labels
General
Statistics
Editor View
Table View
Initial Z Rotation
0.00
Conveyor Section Editor
section 1
Copy of section 1
Section Name
Copy of section 1
Type
Straight
Straight
Curved
Length
Angle
90.00
Radius
5.00
Rise
0.00
Apply
OK
Cancel



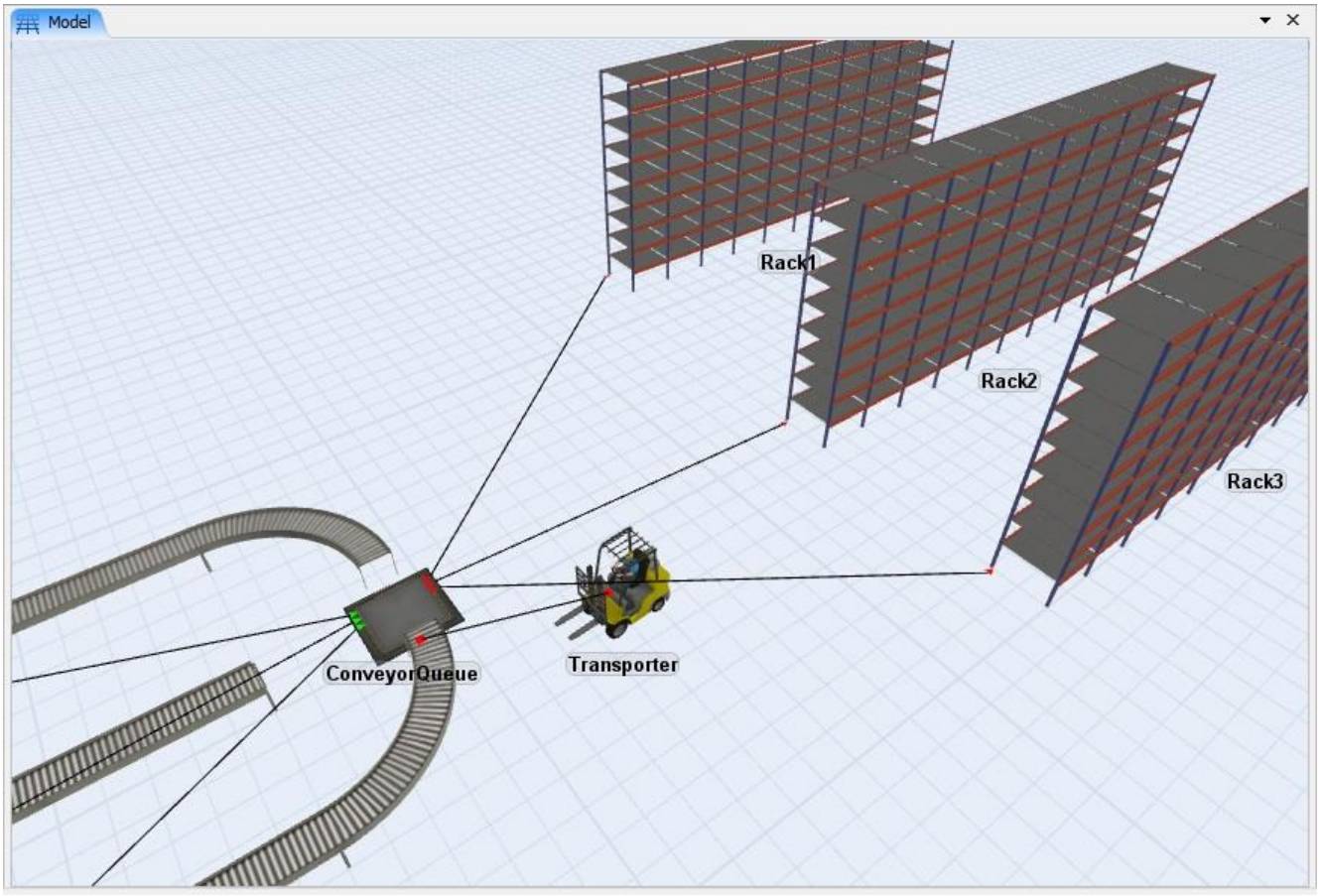
Step 3: Delete the sink

- Highlight the *Sink* and press the **Delete** key.

When an object is deleted, all port connections to and from that object are deleted as well. Beware that this may affect the port numbering of those objects that were connected to the deleted object.

Step 4: Create three Racks

- Create three Racks, place them to the right of *ConveyorQueue*, and name them *Rack1, Rack2,* and *Rack3*. Place the racks far enough away from the queue to allow the fork truck some travel distance to reach the racks.
- Connect *ConveyorQueue* to *Rack1, Rack2,* and *Rack3* (A key).

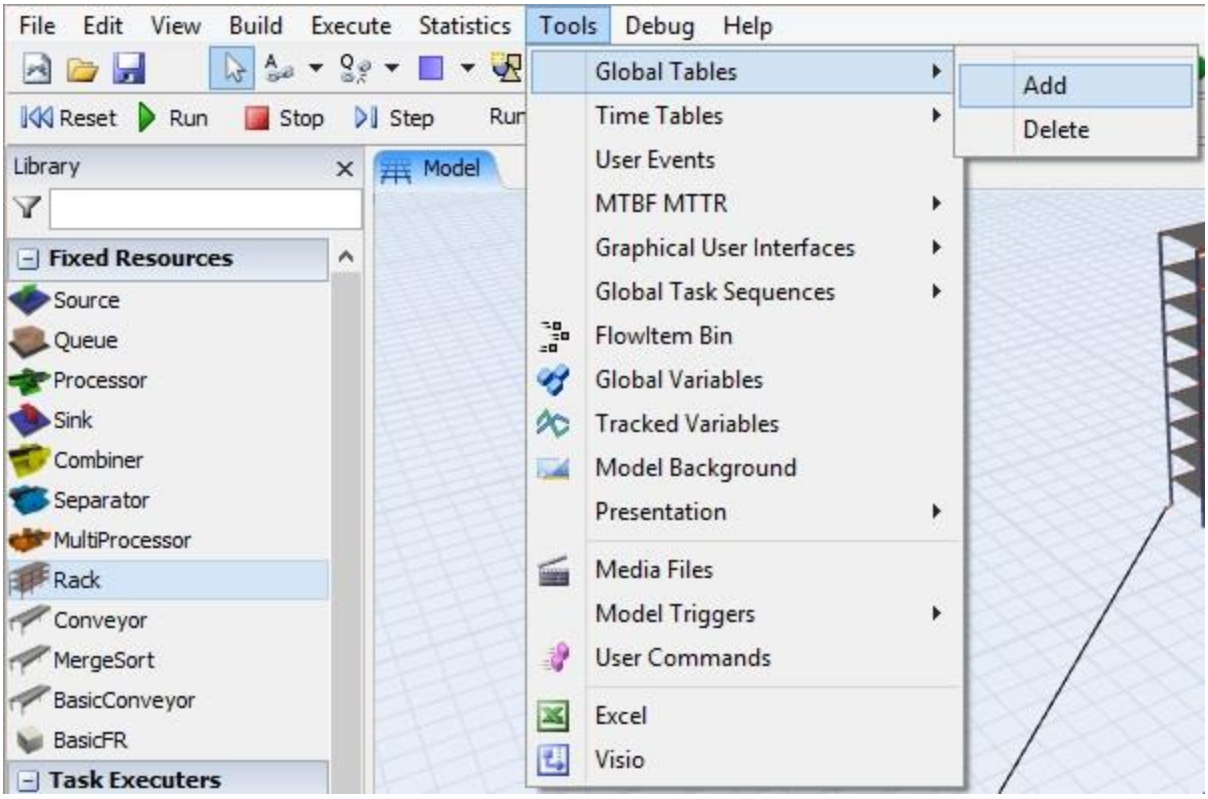


Step 5: Create a Global Table to Control Flowitem Routing

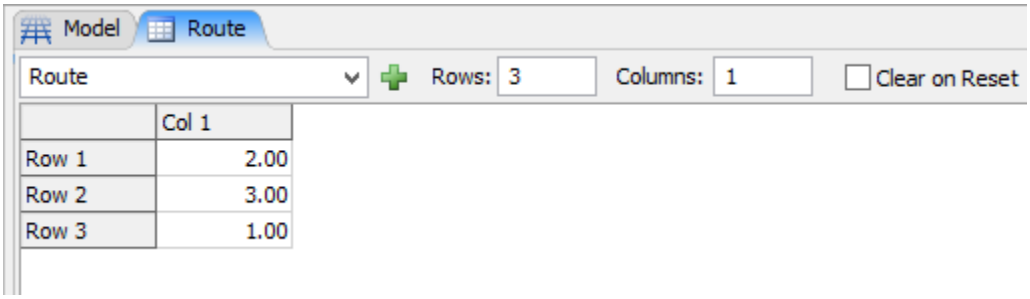
The next step is to set up a global table that will be used to reference which rack each flowitem will be sent to (or more accurately stated, which output port of the conveyor queue the flowitems will be sent through). It is assumed that **output port 1** was connected to *Rack1*, **port 2** to *Rack2*, and **port 3** to *Rack3*. If the connections are not in the correct order, you can modify them through the queue's properties window on the General tab in the Ports section.

We will send all itemtype 1s to *Rack2*, all itemtype 2s to *Rack3*, and all itemtype 3s to *Rack1*. Here are the steps to setting up a global table:

- On the main menu, click **Tools**, point to **Global Tables**, and click **Add.**



- Change the **Name** to **Route**.
- Set **Rows** to 3 and **Columns** to 1.
- Double click on the row names (Row 1, Row 2, Row 3) and name the rows **Item1**, **Item2** and **Item3**, then fill in the values which correspond to the output port number (rack number) we want to send the flowitems to.
- Click the **Close** button to apply the changes and close the table.



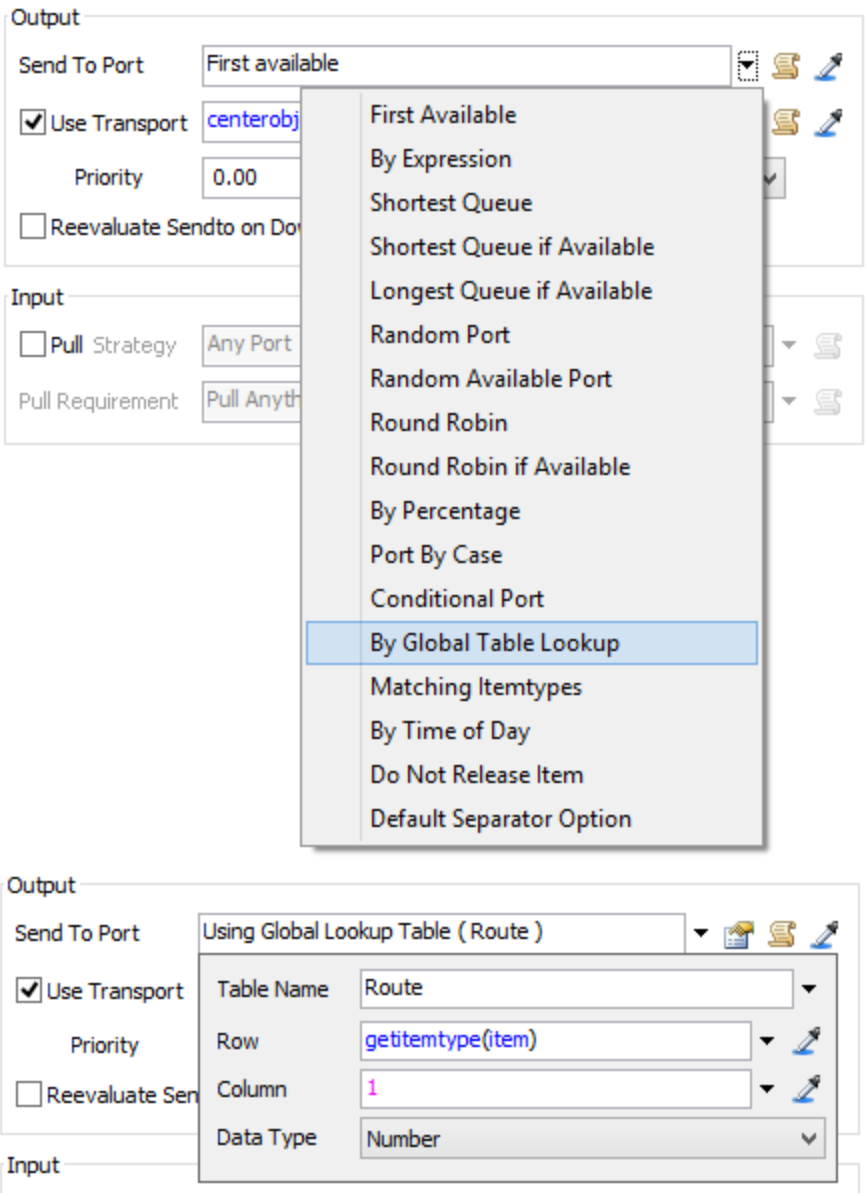
Now that the global table is defined, we can adjust the "Send To Port" option on the queue.

Step 6: Adjusting the Send To Port Option on the ConveyorQueue

You may define the ConveyorQueue's flow and transport options by clicking on the Queue in the 3D view to display its properties in the Quick Properties window.

Alternatively, you can open the object's properties window.

- Double-click on *ConveyorQueue* to bring up its **Properties** window.
- Click the **Flow** tab. Select the option **By Global Table Lookup** from the **Send To Port** list. The code template window will appear. Edit the options to read as follows:



- Click the **OK** button to close the Properties window.

Step 7: Reset, Save, and Run

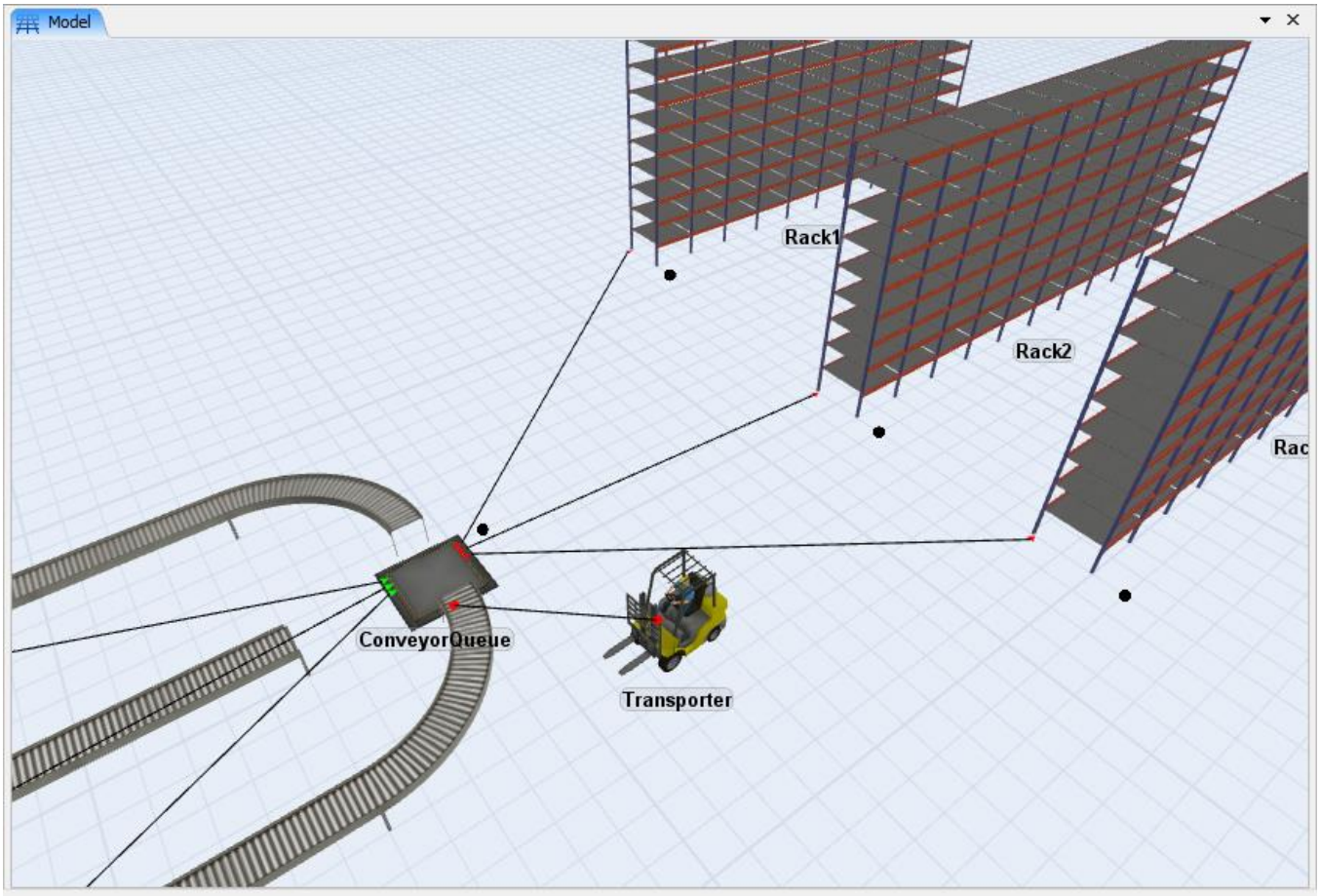
At this point it would be wise to **Reset**, **Save** the model, and then **Run** the model to verify that the changes are working correctly. The model should run with the fork truck transporting flowitems to the racks based on the itemtype definition in the global table.

Step 8: Adding NetworkNodes to develop a path for the Fork Truck

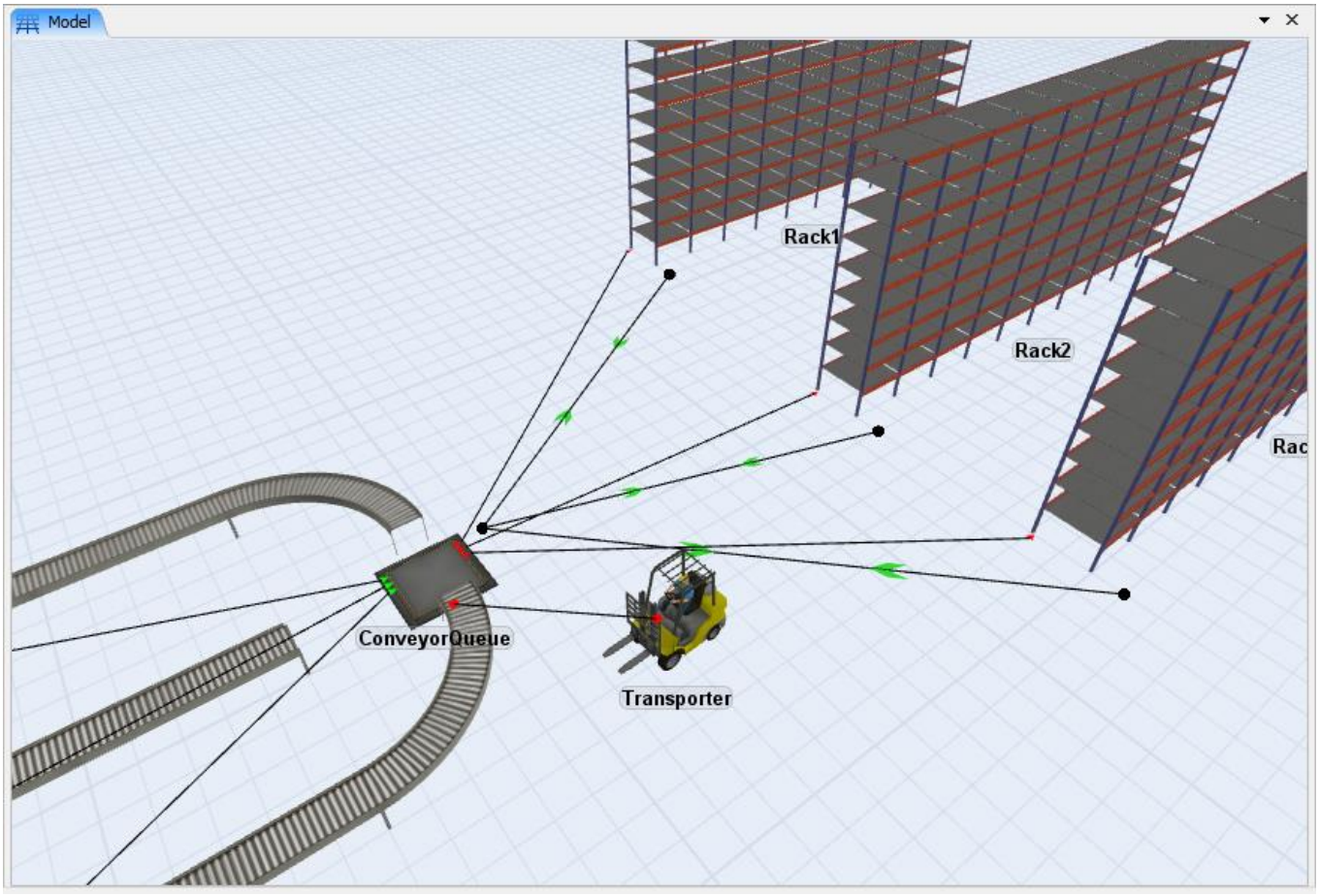
NetworkNode's are used to develop a path network for any task executer object, such as a Transporter, Operator, ASRSvehicle, Crane, etc. In the previous lessons we have used the operator and transporter to transport flowitems around the model. Up to this point we have let the task executer move freely across the model in a direct line between objects. Now we would like to confine the travel of the fork truck to a specific path as it transports flowitems from the conveyor queue to the racks. The following steps are used to set up a simple path.

- Create **NetworkNodes** by dragging them from the library and into the model. Place them near the ConveyorQueue and each of the racks, and name them *NN1, NN2, NN3,* and *NN4.* The nodes will

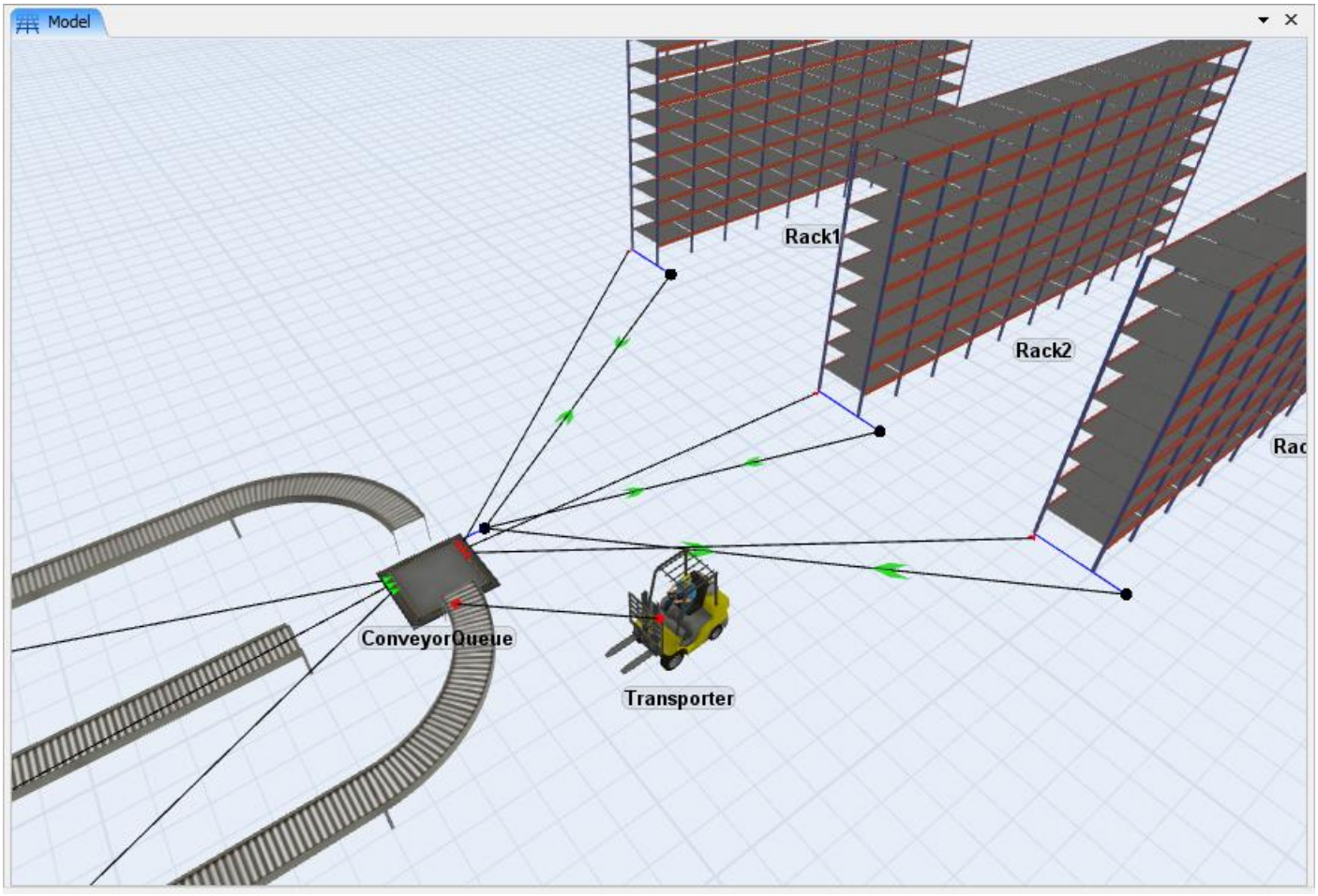
become the pick-up points and drop-off points in the model. You may add additional nodes between these nodes, but it is not necessary.



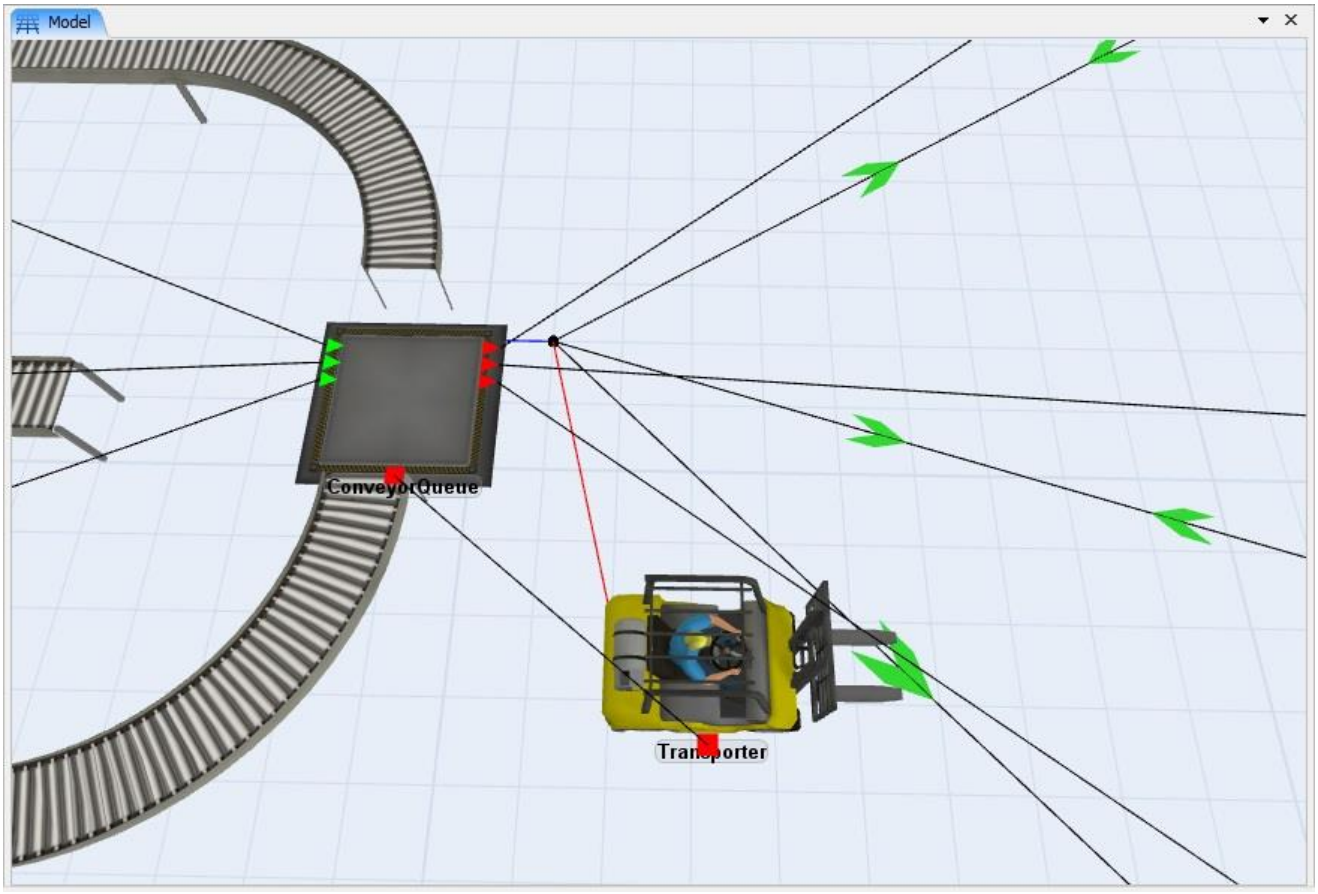
- Connect *NN1* to *NN2, NN3,* and *NN4* (A key). A line will appear after the connection is made with two green boxes along it, indicating that travel is possible in both directions between the two nodes.



- Connect each NetworkNode to the corresponding object (*NN1* to *ConveyorQueue*, *NN2* to *Rack1*, etc.) with the A key. A thin blue line will appear when the connection is made correctly. (If you cannot see the blue line, you may need to move the NetworkNodes)



- The last step is to connect the fork truck to the node network. In order for the fork truck to know that it has to use the path, it must be connected to one of the NetworkNodes in the path network. Connect the *Transporter* to *NN1* (A key). This node now also becomes the starting point for the fork truck when you reset the model.

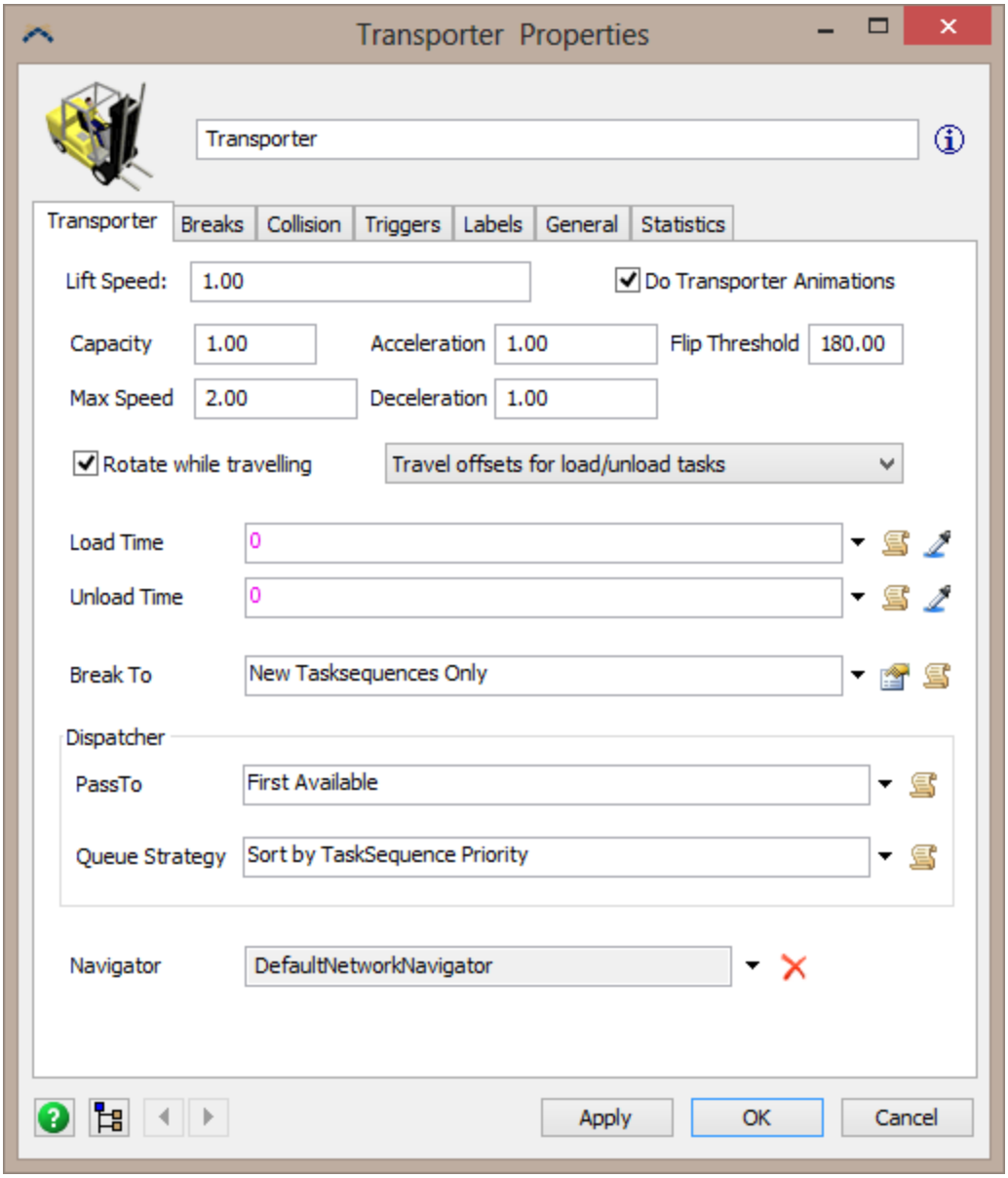


Step 9: Reset, save, and run the model

Now it would be a good idea to **Reset**, **Save**, and then **Run** the model to make sure the fork truck is using the network paths.

A word about offsets

As the model runs, you will notice that the fork truck will travel off the NetworkNode when it picks up or drops off a flowitem. This is a result of having the "Travel offsets for load/unload tasks" selected in the fork truck's properties.

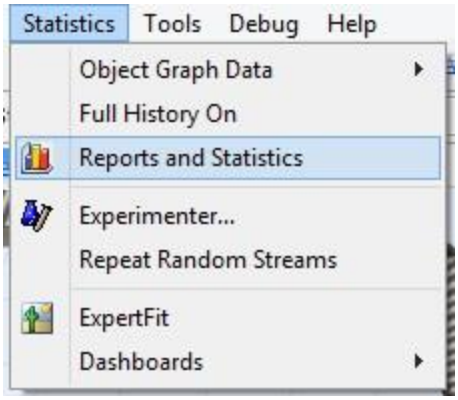


Offsets are used by the fork truck to locate where the flowitem needs to be picked up or dropped off in the object. This allows the fork truck to travel into the queue and pick up the box, and travel to the specific cell in the rack to drop off the box. To force the fork truck to stay at the NetworkNode and not to travel off the path network, select "Do not travel offsets for load/unload tasks" from the drop down picklist found below the field entitled Deceleration.

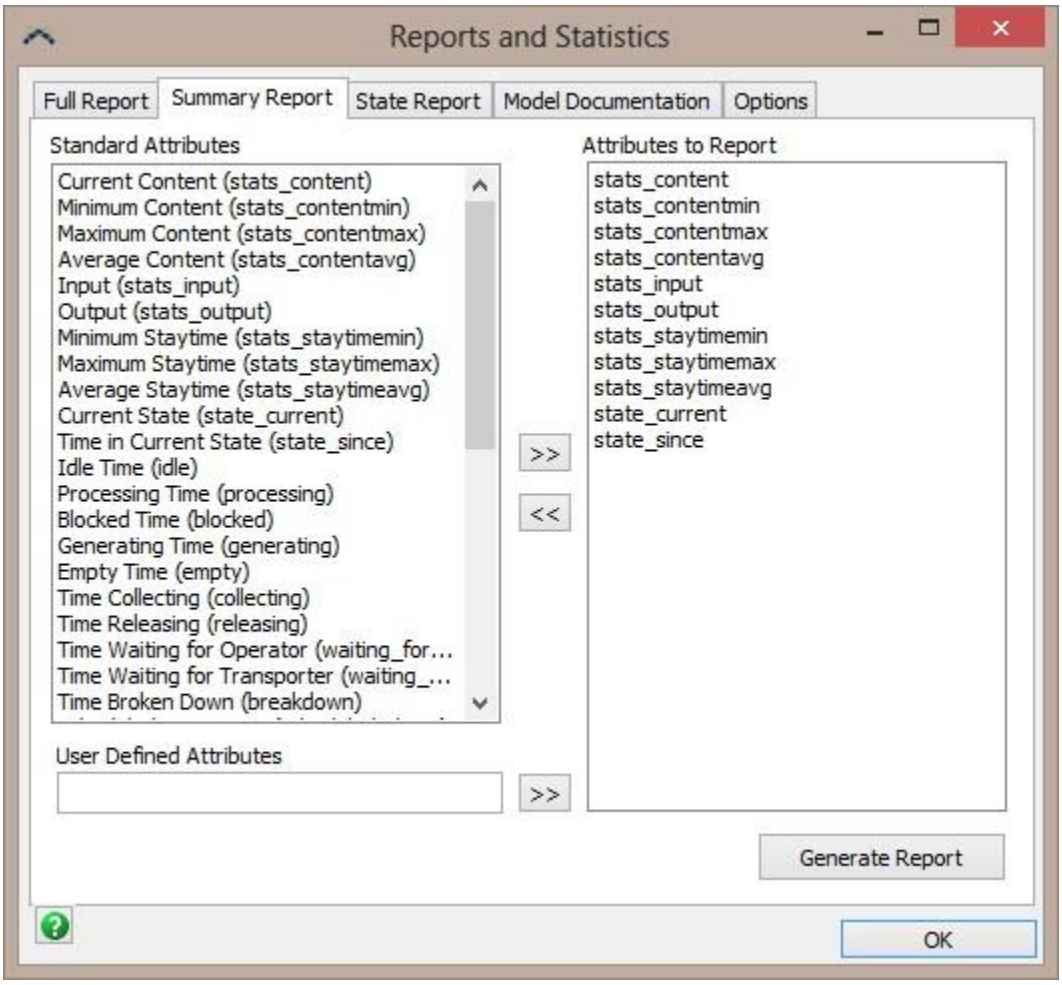
Path networks automatically use Dijkstra's algorithm to determine the shortest distance to travel between any two nodes in the network.

Step 10: Using reports to view output results

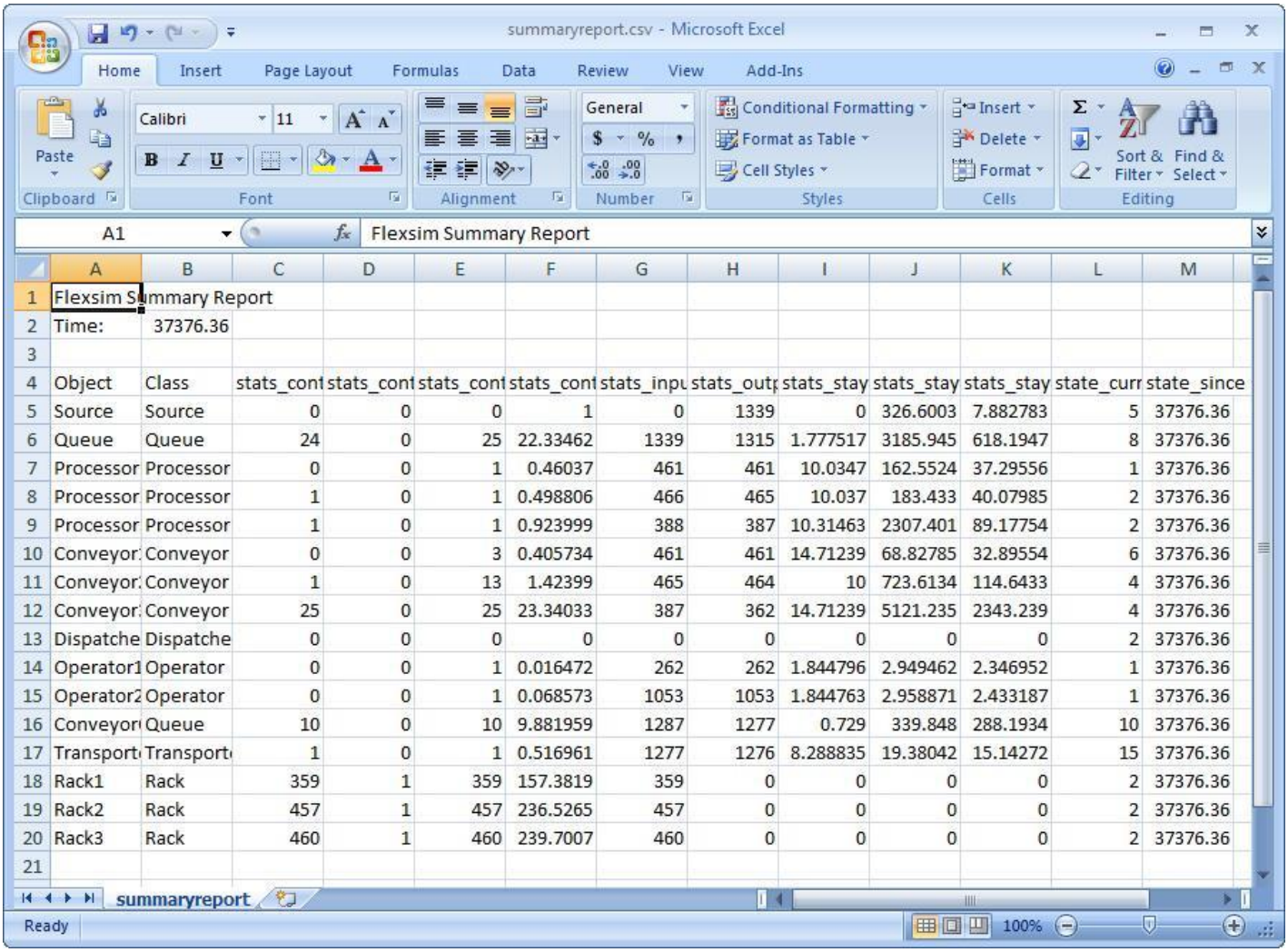
To view summary results of the simulation after having run the model for a period of time, select the main menu option **Statistics** > **Reports and Statistics**.



Go to the **Summary Report** tab of the Reports and Statistics dialog window.

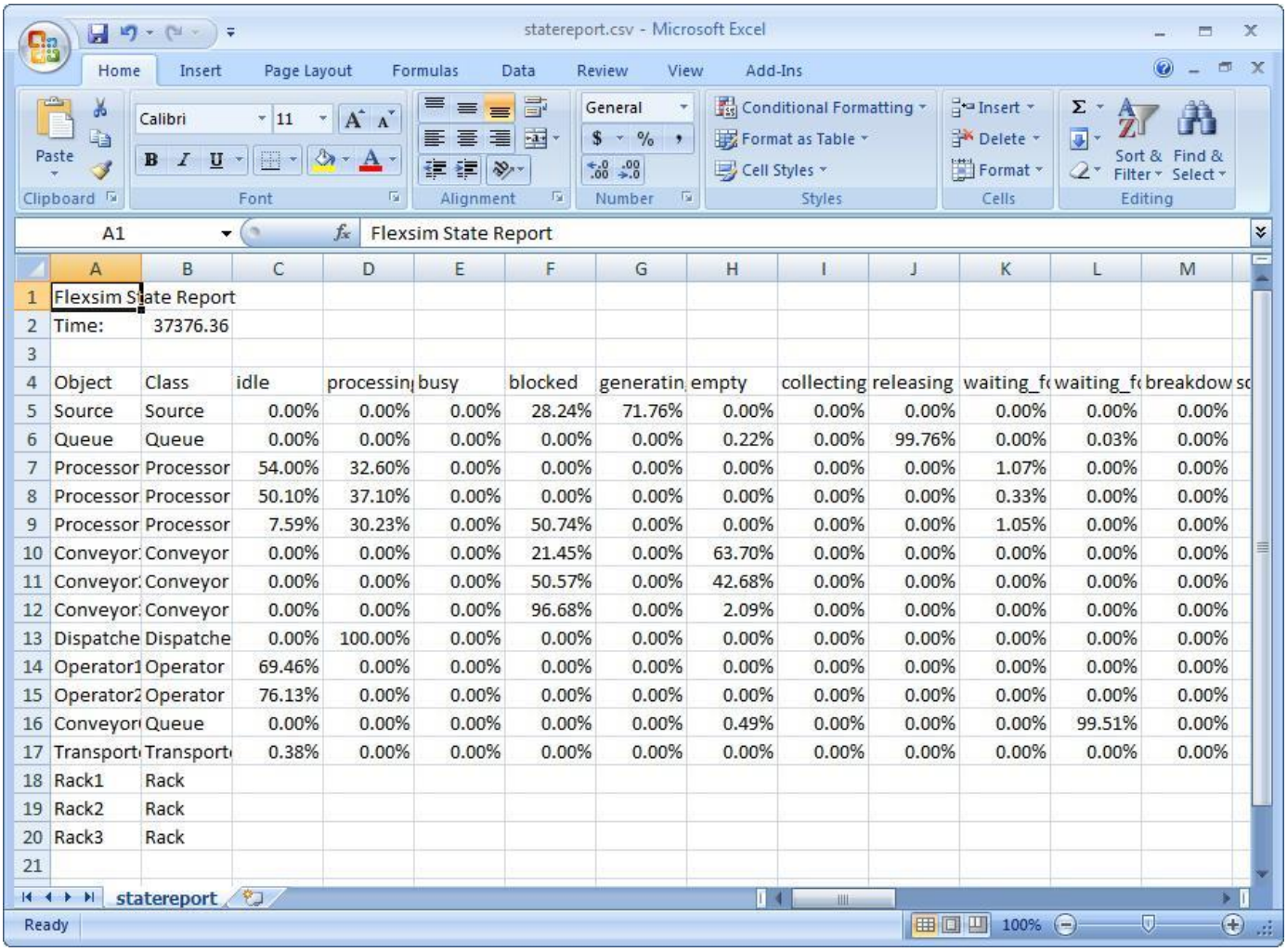


To generate a basic report, press the **Generate Report** button. If you have any other attributes you would like reported, you can add them using the interface provided. The report will be exported to a csv file and automatically displayed in Excel or whichever default program is set up to open csv files on your computer.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Flexsim Summary Report												
2	Time:	37376.36											
3													
4	Object	Class	stats_cont	stats_cont	stats_cont	stats_cont	stats_inpu	stats_outp	stats_stay	stats_stay	stats_stay	state_curr	state_since
5	Source	Source	0	0	0	1	0	1339	0	326.6003	7.882783	5	37376.36
6	Queue	Queue	24	0	25	22.33462	1339	1315	1.777517	3185.945	618.1947	8	37376.36
7	Processor	Processor	0	0	1	0.46037	461	461	10.0347	162.5524	37.29556	1	37376.36
8	Processor	Processor	1	0	1	0.498806	466	465	10.037	183.433	40.07985	2	37376.36
9	Processor	Processor	1	0	1	0.923999	388	387	10.31463	2307.401	89.17754	2	37376.36
10	Conveyor	Conveyor	0	0	3	0.405734	461	461	14.71239	68.82785	32.89554	6	37376.36
11	Conveyor	Conveyor	1	0	13	1.42399	465	464	10	723.6134	114.6433	4	37376.36
12	Conveyor	Conveyor	25	0	25	23.34033	387	362	14.71239	5121.235	2343.239	4	37376.36
13	Dispatche	Dispatche	0	0	0	0	0	0	0	0	0	2	37376.36
14	Operator1	Operator	0	0	1	0.016472	262	262	1.844796	2.949462	2.346952	1	37376.36
15	Operator2	Operator	0	0	1	0.068573	1053	1053	1.844763	2.958871	2.433187	1	37376.36
16	Conveyor	Queue	10	0	10	9.881959	1287	1277	0.729	339.848	288.1934	10	37376.36
17	Transport	Transport	1	0	1	0.516961	1277	1276	8.288835	19.38042	15.14272	15	37376.36
18	Rack1	Rack	359	1	359	157.3819	359	0	0	0	0	2	37376.36
19	Rack2	Rack	457	1	457	236.5265	457	0	0	0	0	2	37376.36
20	Rack3	Rack	460	1	460	239.7007	460	0	0	0	0	2	37376.36
21													

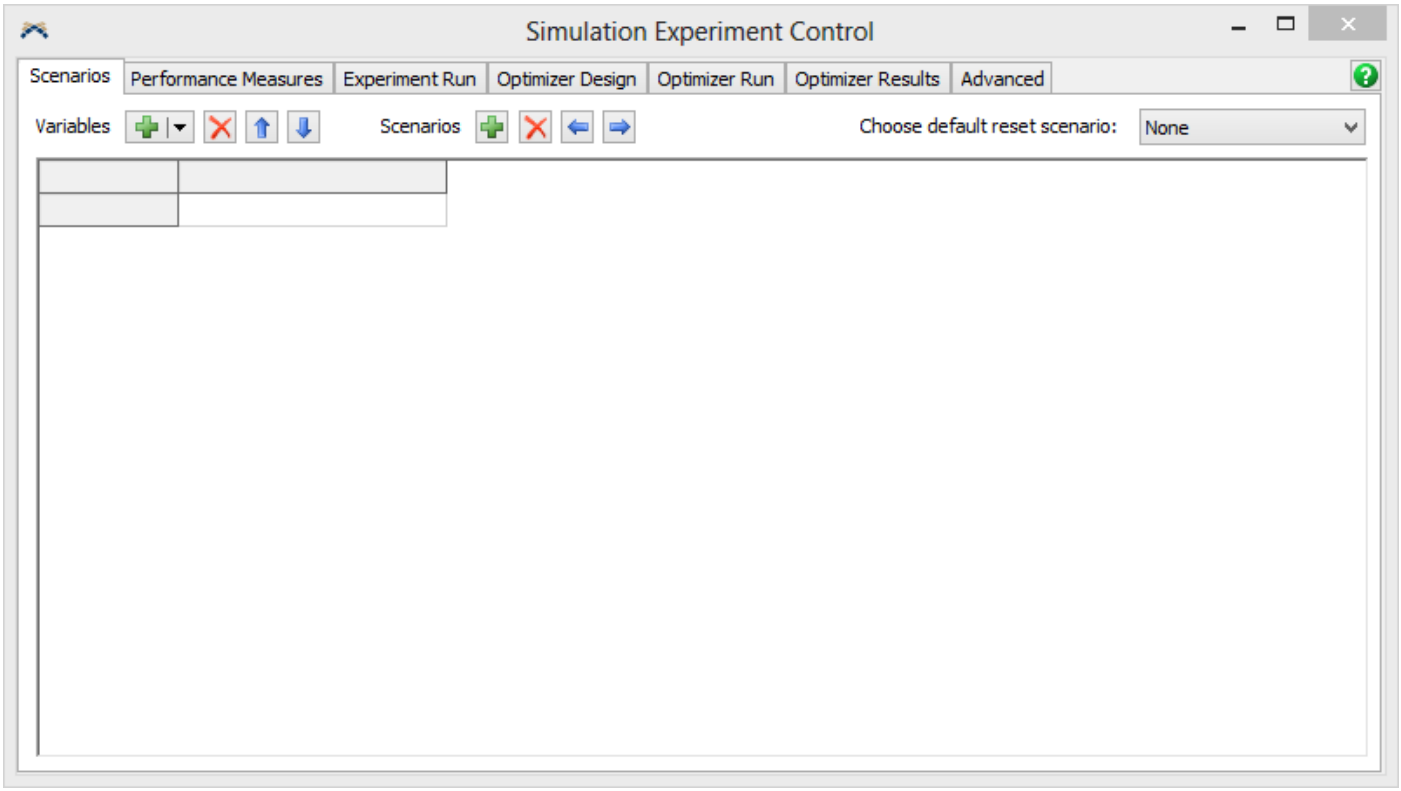
To create a state report, go to the **State Report** tab of the Reports and Statistics dialog window, and press **Generate Report**.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Flexsim State Report												
2	Time:	37376.36											
3													
4	Object	Class	idle	processin	busy	blocked	generatin	empty	collecting	releasing	waiting_f	waiting_f	breakdow
5	Source	Source	0.00%	0.00%	0.00%	28.24%	71.76%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
6	Queue	Queue	0.00%	0.00%	0.00%	0.00%	0.00%	0.22%	0.00%	99.76%	0.00%	0.03%	0.00%
7	Processor	Processor	54.00%	32.60%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	1.07%	0.00%	0.00%
8	Processor	Processor	50.10%	37.10%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.33%	0.00%	0.00%
9	Processor	Processor	7.59%	30.23%	0.00%	50.74%	0.00%	0.00%	0.00%	0.00%	1.05%	0.00%	0.00%
10	Conveyor	Conveyor	0.00%	0.00%	0.00%	21.45%	0.00%	63.70%	0.00%	0.00%	0.00%	0.00%	0.00%
11	Conveyor	Conveyor	0.00%	0.00%	0.00%	50.57%	0.00%	42.68%	0.00%	0.00%	0.00%	0.00%	0.00%
12	Conveyor	Conveyor	0.00%	0.00%	0.00%	96.68%	0.00%	2.09%	0.00%	0.00%	0.00%	0.00%	0.00%
13	Dispatche	Dispatche	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
14	Operator1	Operator	69.46%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
15	Operator2	Operator	76.13%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
16	Conveyor	Queue	0.00%	0.00%	0.00%	0.00%	0.00%	0.49%	0.00%	0.00%	0.00%	99.51%	0.00%
17	Transport	Transport	0.38%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
18	Rack1	Rack											
19	Rack2	Rack											
20	Rack3	Rack											
21													

Step 11: Running multiple runs of your model using the Experimenter

To access the Experimenter in FlexSim, select the main menu option **Statistics** > **Experimenter**. The Simulation Experiment Control window will appear.



The Simulation Experiment Control window is used to run multiple replications of a given model, and to run multiple scenarios of a model. When running multiple scenarios, you can declare a number of experiment variables, and then specify the values you want these variables to be set to for each of the scenarios you want to run. Confidence intervals are calculated and displayed for each of the performance measures you define on the Performance Measures tab. For more information on the experimenter, refer to the Experimenter section of the help documentation.

This completes lesson 3. Congratulations!

Labels Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

This tutorial will introduce you to using Labels in your model. Labels can be used to store information in your objects and items that can be accessed at anytime. As a couple of examples, labels can be useful to specify the downstream flow of your items, or they can be used to store financial data about an object.

What You Will Learn

- How to create and access labels
- How you can use labels to change the way your model functions

New Features

- You will be introduced to Pull Requirements
- You will be introduced to the FlowItem Bin

Approximate Time to Complete this Lesson

This lesson should take about 20- 30 minutes to complete.

Model Overview

In this model we will create a simple model that creates items with three different itemtypes. We will then track how many items we create of each itemtype and then upon processing the item, we will modify the size of our items.

Click here for the Step-By-Step Tutorial.

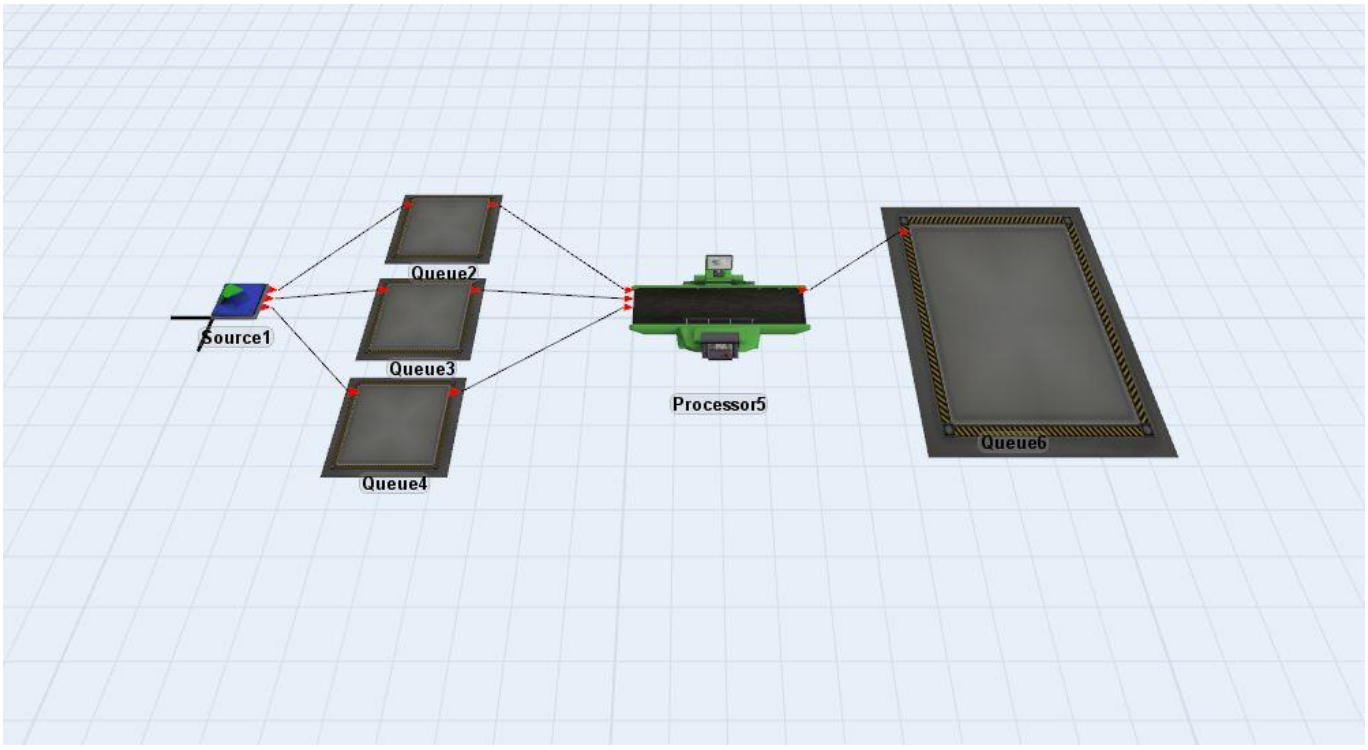
Step-By-Step Model Construction

Building Labels Model

Begin a new model by clicking the button on the toolbar. Click **OK** on the Model Units window – we will use the default units for our model.

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.



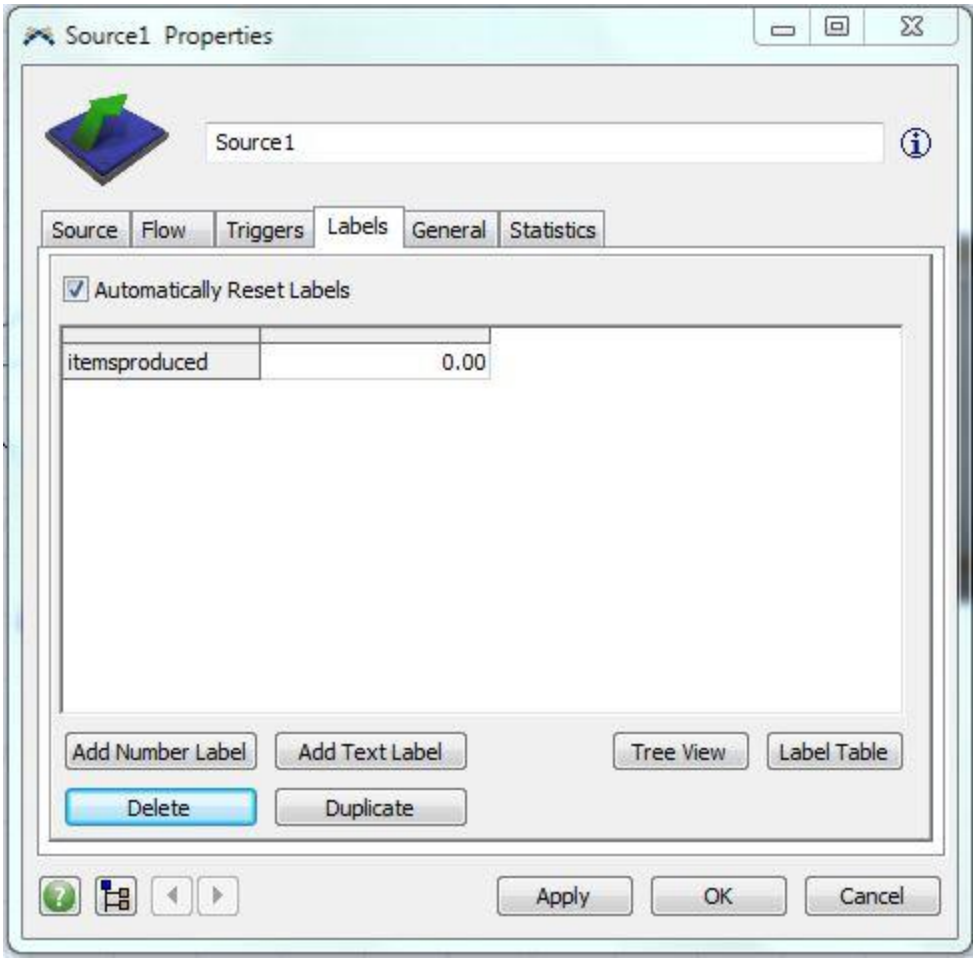
Connect all of the objects as shown:

- Connect *Source1* to *Queue2 Queue3* and *Queue4*
- Connect *Queue2*, *Queue3* and *Queue4* to *Processor5*
- Connect *Processor5* to *Queue6*

Step 2: Setup the Source Properties

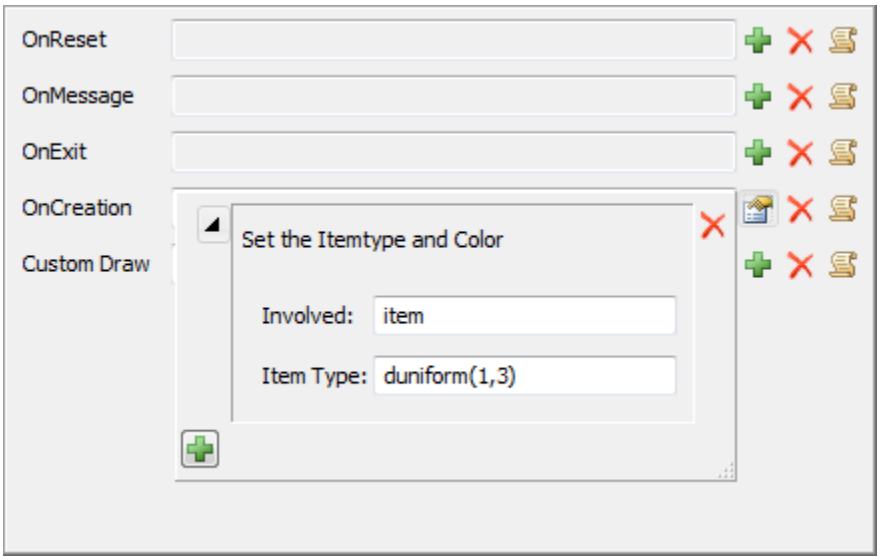
All objects can store their own Labels. The Labels tab will allow you to create, view and manipulate the labels on that object.

- Double-click on the *Source* to open its properties window.
- Go to the **Labels** tab.
- Check **Automatically Reset Labels**. This will cause the labels to reset to the value you give it when you reset the model.
- Click **Add Number Label** and double click *"newlabelname"* to rename the label *"itemsproduced"*.



The *Source* will create items with 3 itemtypes. Each itemtype will be assigned a specific color.

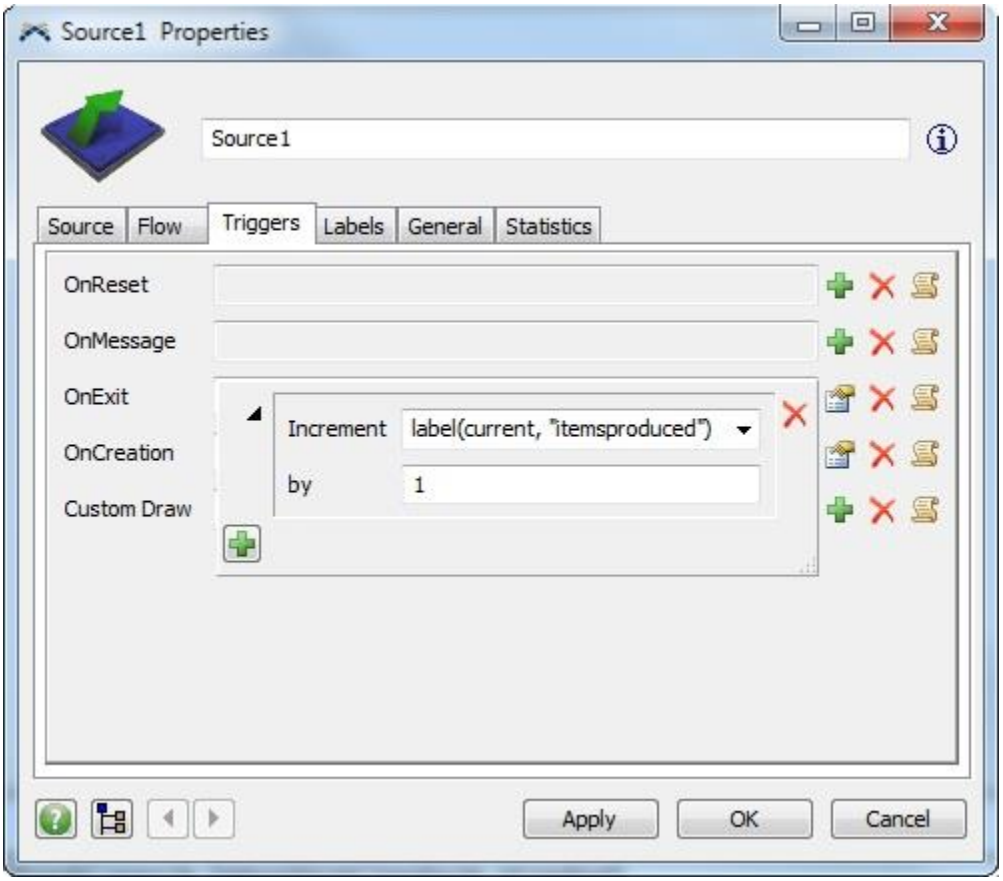
- Go to the **Triggers** tab.
- Click the add button for the **OnCreation** trigger, and select the *Set Item Type and Color* pick option.
- We will be using 3 itemtypes, so keep the **Involved** and **Item Type** at their default values.



We also want to increment the *"itemsproduced"* label to keep track of how many items have been produced by the source.

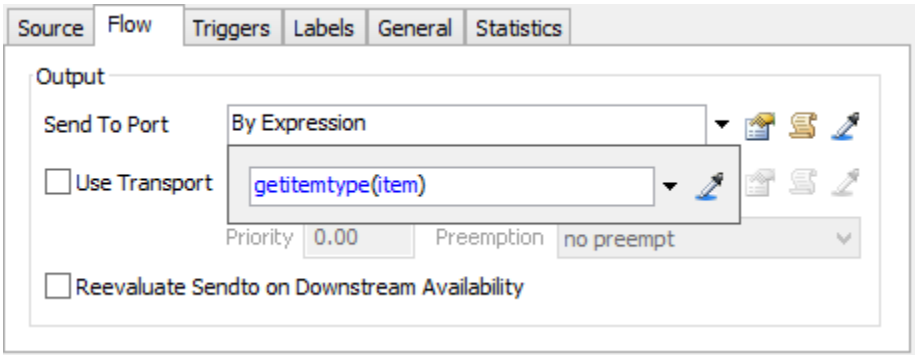
- Go to the **Triggers** tab.

- Click the add button for the **OnExit** trigger, and select the **Increment Value** pick option.
- In the edit box for **Increment** replace *label(item, "labelname")* with *label(current, "itemsproduced")*.
- Leave the **By** at *1*. This will increment the value of the *"itemsproduced"* label by 1 on each time an item leaves the Source.



We will also assign items of each itemtype to go to a specific queue based on the itemtype.

- Go to the **Flow** tab.
- Choose **By Expression** in the **Send to Port** option. Leave it at the default *getitemtype(item)*.



- Click Apply to apply the changes you made to the *Source*.

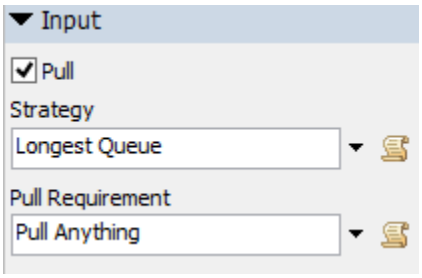
Reset and run your model. If you go back to the **Labels** tab of the source and watch your *"itemproduced"* label, it will update the value as each new flow item is created.

Close the properties window of the Source. It may ask you if you want to "Update Label Reset Values?". Click No. (You want the labels to reset to zero when you reset the model.)

Step 3: Setup the Processor Properties

We want the Processor to grab items from the Queue that has the most items waiting in, or the **Longest Queue**. Since each of the Queues do not know how many items are in the other Queues, it is works best to have the Processor determine which Queue it will pull an item from.

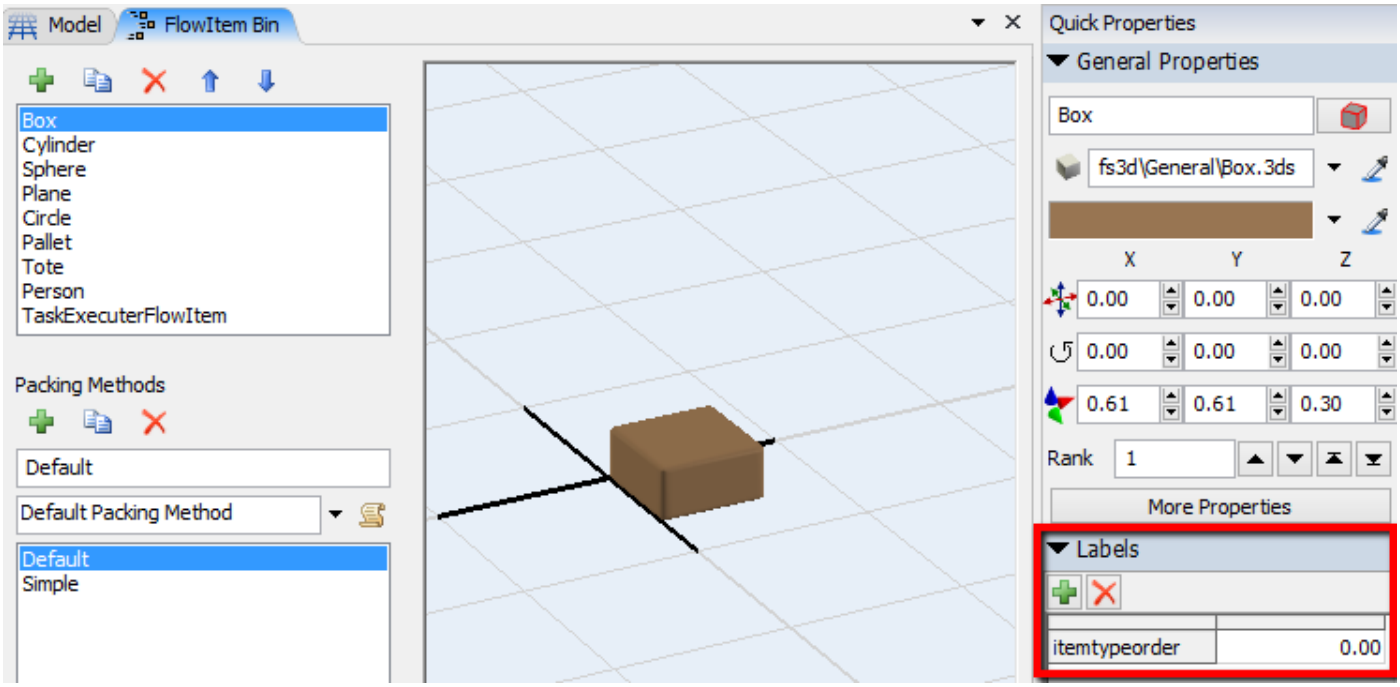
- Click on the *Processor* to open its properties in the Quick Properties window.
- Click the arrow next to the **Input** section to expand it.
- Check the **Pull** option and set the **Pull Strategy** to *Longest Queue*.



Step 4: Customizing Flowitems

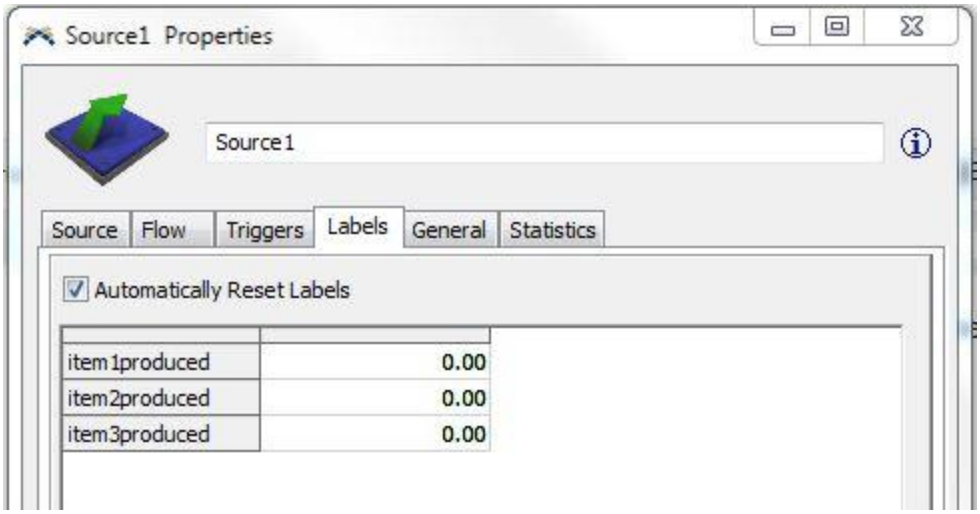
We are going to store information on all the flowitems so that they can be accessed throughout the model.

- Open the **Flowitem Bin** by going to **Tools > Flowitem Bin**
- With **Box** selected click the ✚ under the Labels section of the Quick Properties window and add a number label.
- Double click *"LabelName"* to rename the label *"itemtypeorder"*.



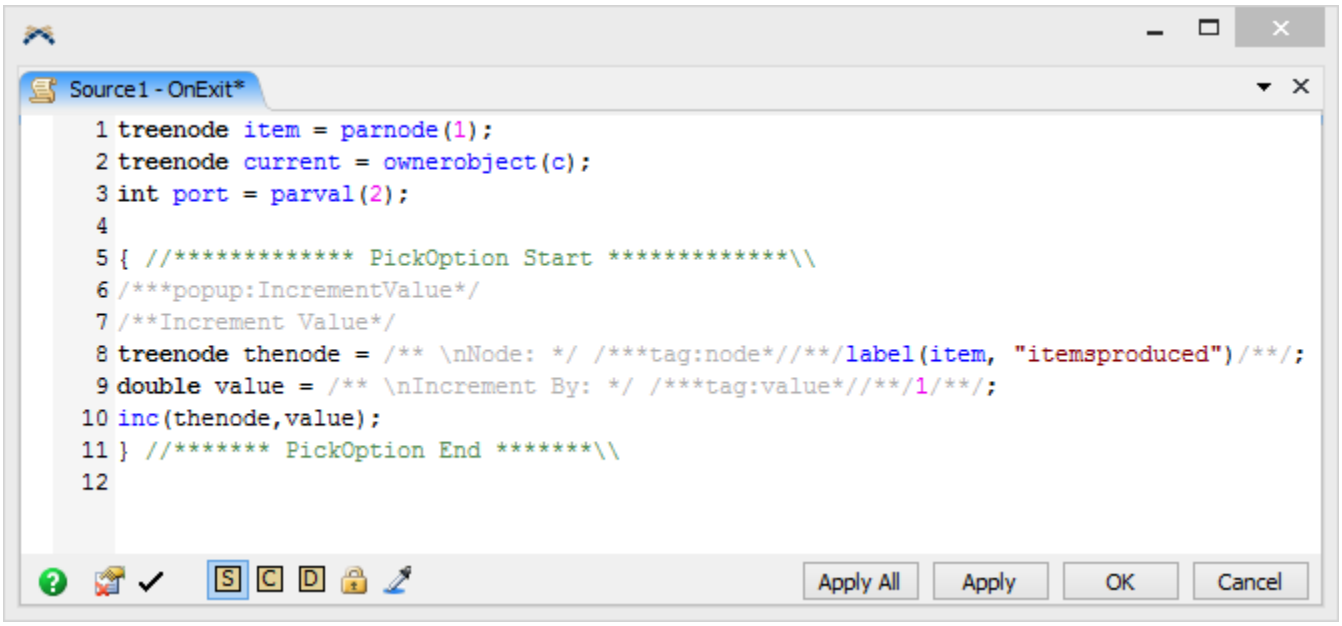
We will now have the source increment the *"itemtypeorder"* label on each flowitem for each specific type of item. This will allow us to know in what order the flowitem was created.

- Open the properties window for the *Source*.
- Go to the **Labels** tab.
- Change the label *"itemsproduced"* to *"item1produced"*.
- Select the *"item1produced"* label and hit the **Duplicate** button twice.
- Rename the two new labels to *"item2produced"* and *"item3produced"*.

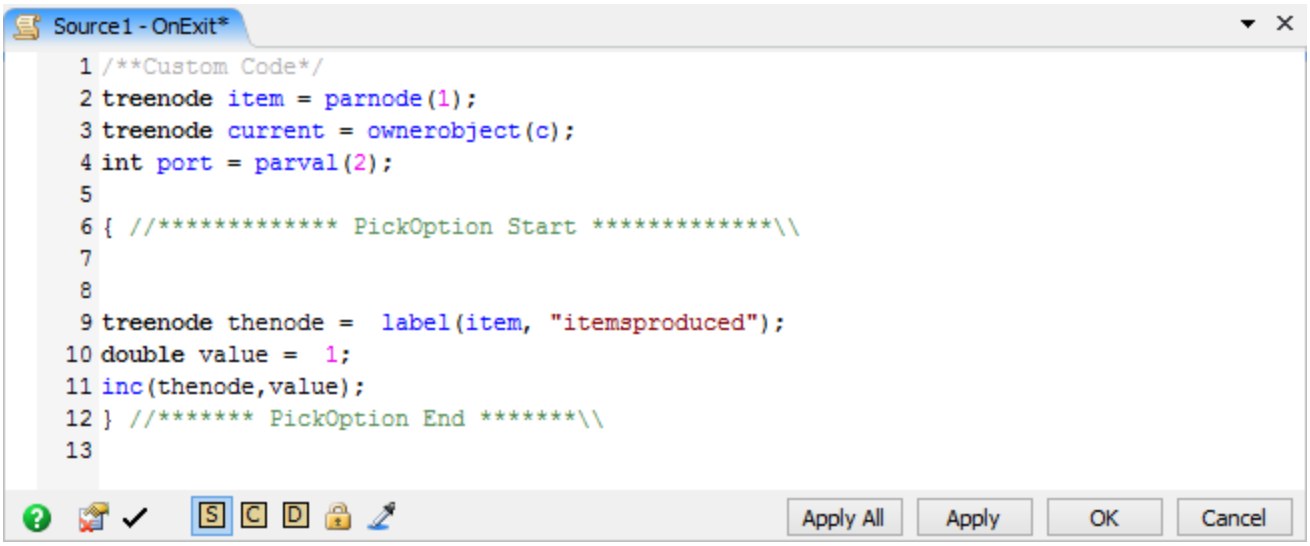


In order to increment each these labels correctly, we need to customize the OnExit trigger. This will require editing the code manually, but don't worry – writing code is easy!

- Go into the Triggers tab, and click on the button, located to the left corner of the OnExit Trigger, to open the **Code Editor**.



- Click on the remove template button to get rid of the comments.
- Feel free to delete the "PickOption Start/End" lines.



- We want this *"increment"* function to increment each label individually. Write an "if" statement matching each itemtype to its own label. We will also set the new label's value to the flowitem.

```
if (getitemtype(item) == 1)
{
     int value = inc(label(current, "item1produced"), 1);
     setlabelnum(item, "itemtypeorder", value);
}
```

- Write the same statements for itemtype 2 and 3. The code should look like this when you are done:

```
/**Custom Code*/
treenode item = parnode(1);
treenode current = ownerobject(c);
int port = parval(2);

if (getitemtype(item) == 1)
{
    int value = inc(label(current, "item1produced"), 1);
    setlabelnum(item, "itemtypeorder", value);
}
if (getitemtype(item) == 2)
{
    int value = inc(label(current, "item2produced"), 1);
    setlabelnum(item, "itemtypeorder", value);
}
if (getitemtype(item) == 3)
{
    int value = inc(label(current, "item3produced"), 1);
    setlabelnum(item, "itemtypeorder", value);
}
```

- Click **OK** to apply and close the Code window.
- Click **OK** to apply and close the Properties window.

Step 5: Process the Flowitems

In this step, we will change the way the *Processor* processes the boxes.

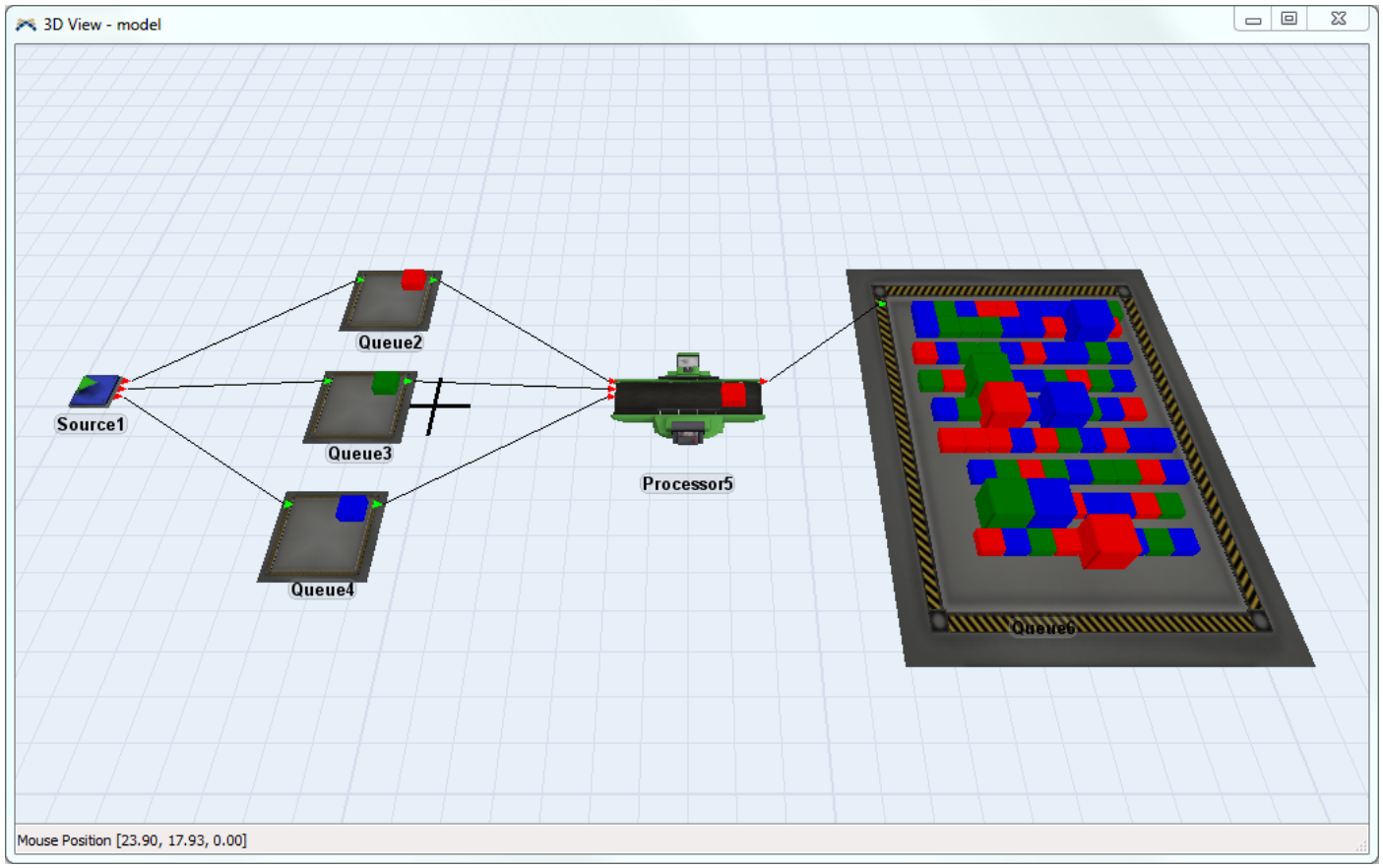
- Open the *Processor's* Properties window.
- Go to the *Triggers* tab.
- Click the code edit button next to the *OnProcessFinish* trigger.
- Write an if statement that will only allow every tenth item of each type to be set to a specific size:

```
if (getlabelnum(item, "itemtypeorder") % 10 == 0)
{
     setsize(item,1,1,1);
}
```

Note: You don't need to do this two more times, because it applies to any flowitem with the *"itemtypeorder"* label.

- Click **OK** on the Edit Code window as well as the Properties window.

Reset and **Run** your model. Your model should begin to look something like this:



This completes the the Labels tutorial. Congratulations!

Global Modeling Tools Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

This lesson will introduce the basic concepts of global modeling tools that can be used throughout your model. These properties can simplify views and enable dynamic changes in your model. We will also look at ways to use a Combiner and Separator in order to better model situations that may happen in real life.

Note: For more information on global modeling tools, see Modeling Tools.

What You Will Learn

- How to create and access Global Tables
- How to create and access Global Variables and Global Macros
- How to write a simple User Command for use in your model
- How to use a Combiner and Separator object

New Objects

In this lesson you will be introduced to the Combiner and Separator objects.

Approximate Time to Complete this Lesson

This lesson should take about 45-60 minutes to complete.

Model Overview

In this model we're going to look at some different ways of accomplishing tasks as opposed to how they have been done in previous lessons. Both methods give the same result, but some users prefer one way over the other.

We'll also take a look at Combiners and Separators and how they can be used in your models.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

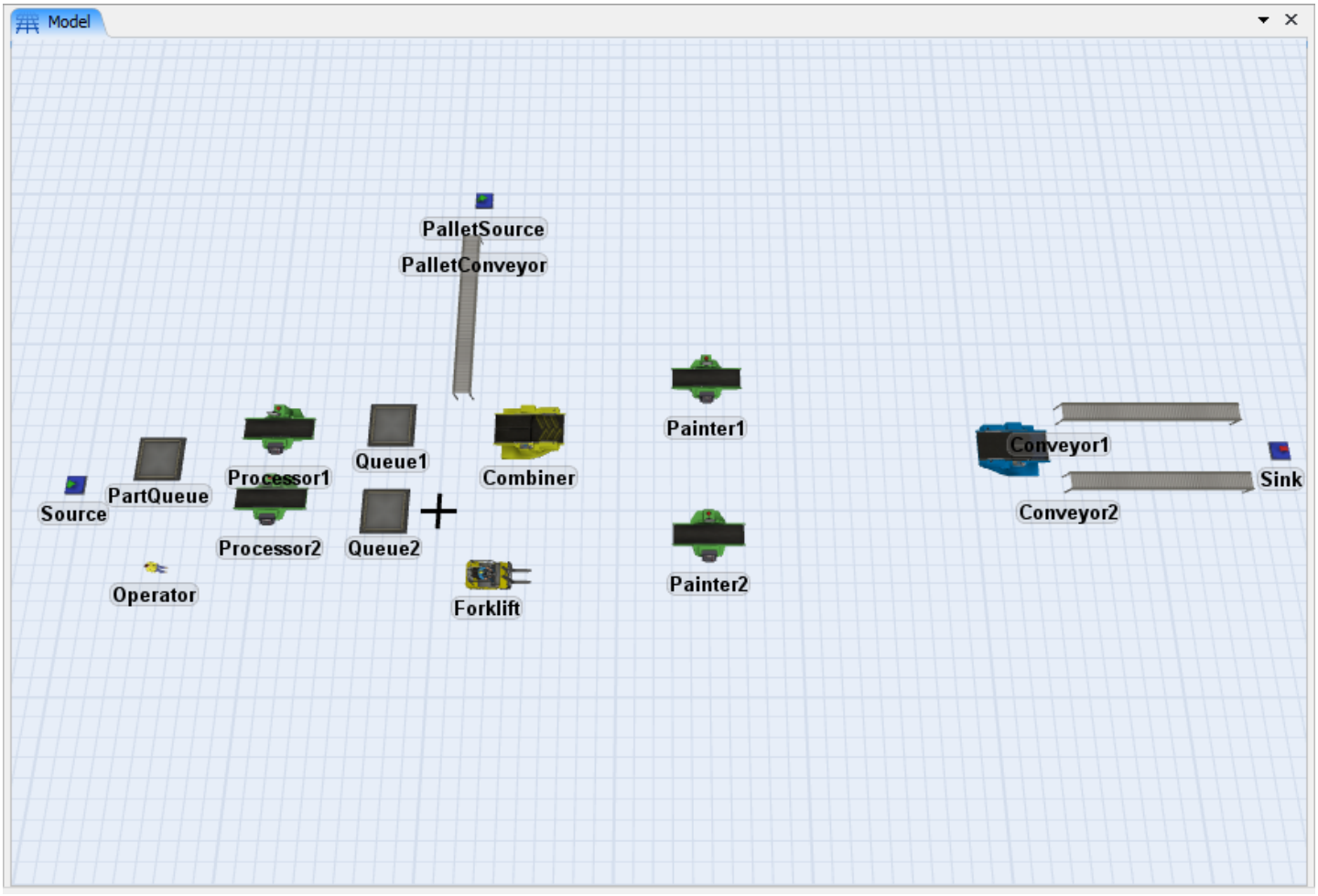
Building Global Modeling Tools Model

Begin a new model by clicking the button on the toolbar. Click **OK** on the Model Units window; we will use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.
- Rename the objects as shown as we will refer to these objects by name throughout the tutorial.



Step 2: Connect the ports

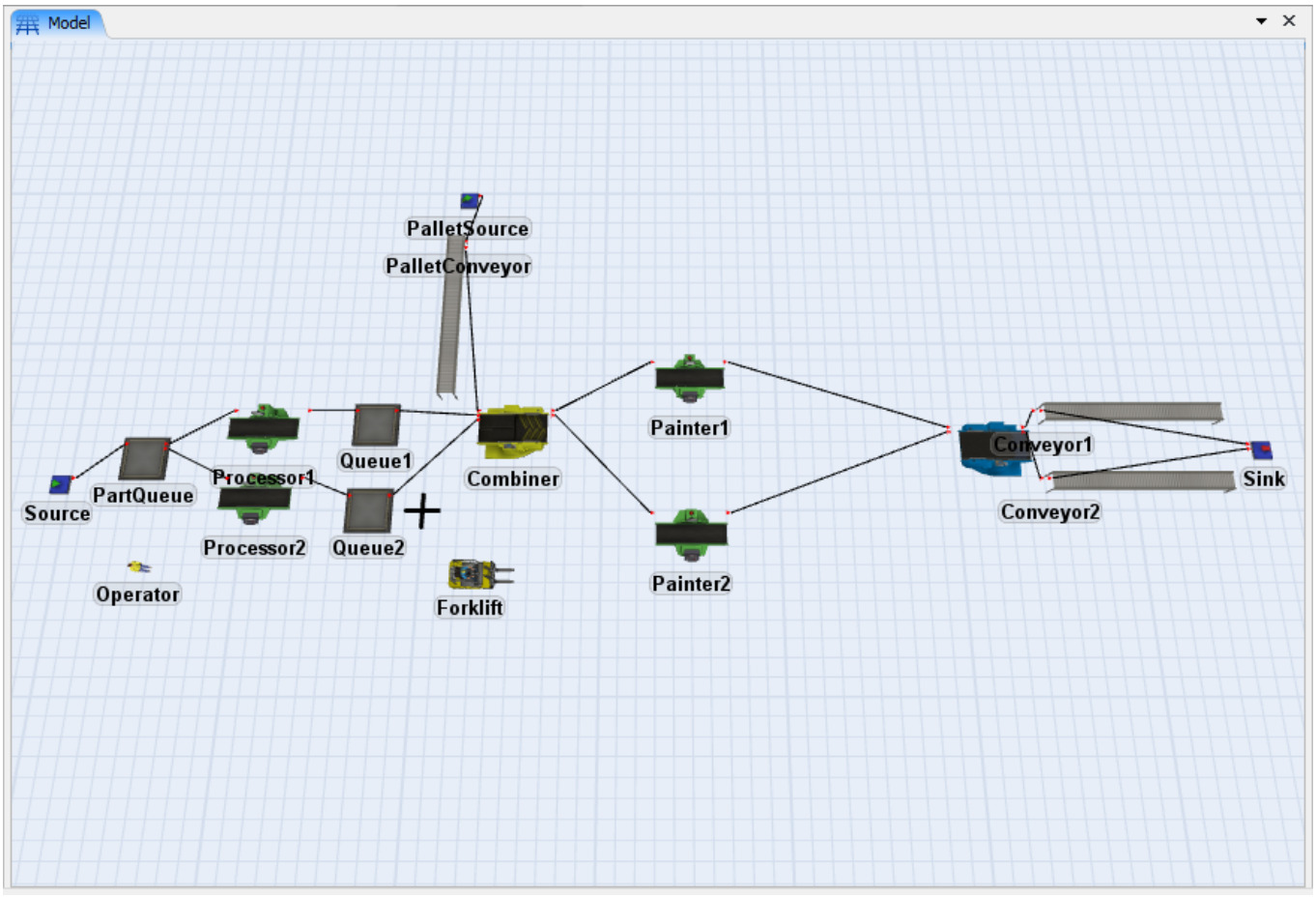
Connect all of the objects as shown below.

- Connect *Source* to *PartQueue.*
- Connect *PartQueue* to *Processor1* and *Processor2*
- Connect *Processor1* to *Queue1* and *Processor2* to *Queue2*
- Connect *PalletSource* to *PalletConveyor*
- Connect *PalletConveyor* to *Combiner*

Note: It is required to have *PalletConveyor* attached to the 1st input port of *Combiner*.

- Connect *Queue1* and *Queue2* to *Combiner*
- Connect *Combiner* to *Painter1* and *Painter2*
- Connect *Painter1* and *Painter2* to *Separator*
- Connect *Separator* to *Conveyor1* and *Conveyor2*
- Connect *Conveyor1* and *Conveyor2* to *Sink*

Notice that we did not make any center connections to *Operator* or *Forklift*. We will tie those in using Global Variables.



Step 3: Set the Source Properties

Set up the source to create two item types.

- Leave the default **Inter-Arrival Time** of *Source* at *exponential(0, 10, 0)*.
- Set the **OnCreation** trigger to **Set Itemtype and Color** with an itemtype of *duniform(1,2)*.
- Click **OK** to apply and close the Properties window.

Step 4: Set the PalletSource Properties

Our pallets are going to act as dummy objects that will hold our other parts as they go through the *Combiner* and *Painter*. Setting the FlowItem Class to a pallet will cause the *Combiner* to neatly stack our other flowitems on top of the pallet item.

Our pallet is also going to be used to determine what items are combined in the *Combiner*. This is specified by setting the itemtype of each pallet. This will be explained later on.

- Set the **FlowItem Class** of *PalletSource* to Pallet (this can be done through the Source's properties under the Source tab or through the Quick Properties).
- To have an infinite supply of pallets, set the **Inter-Arrival Time** to 0.
- Go to the **Triggers** tab and add a trigger to the **OnCreation** of *Set Type, Name or Label*.
- Change the **Type** to *duniform(1,6)*.
- Click **OK** to apply and close the Properties window.

At this point, feel free to try your model out. You should see your flowitems run through the entire model, combining a pallet with one item from *Queue1* and one item from *Queue2* at the *Combiner*, and then separating those items at the *Separator*.

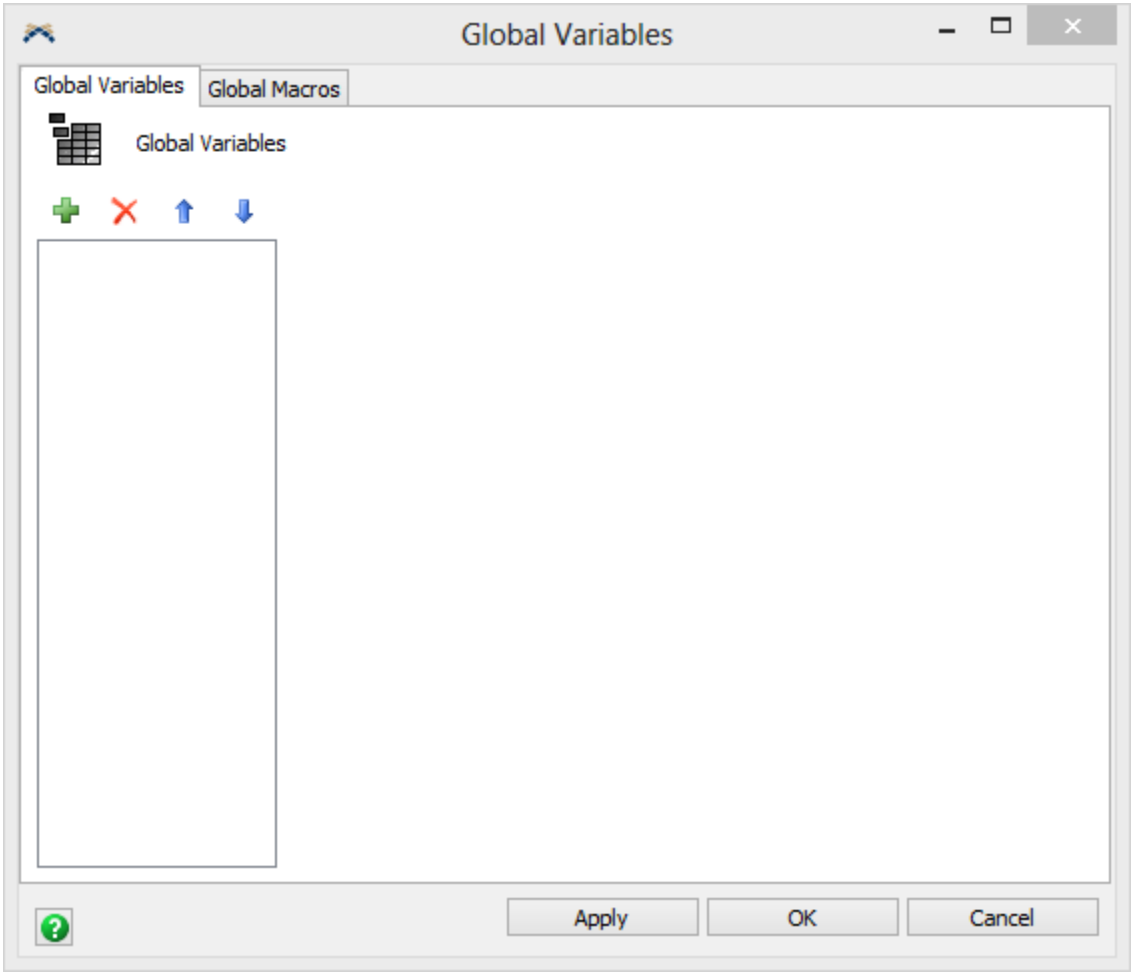
Step 5: Add the Operators

To show another way of connecting objects together, we are not going to connect our operators using center ports. Instead, we're going to create Global Variables that point to our operators.

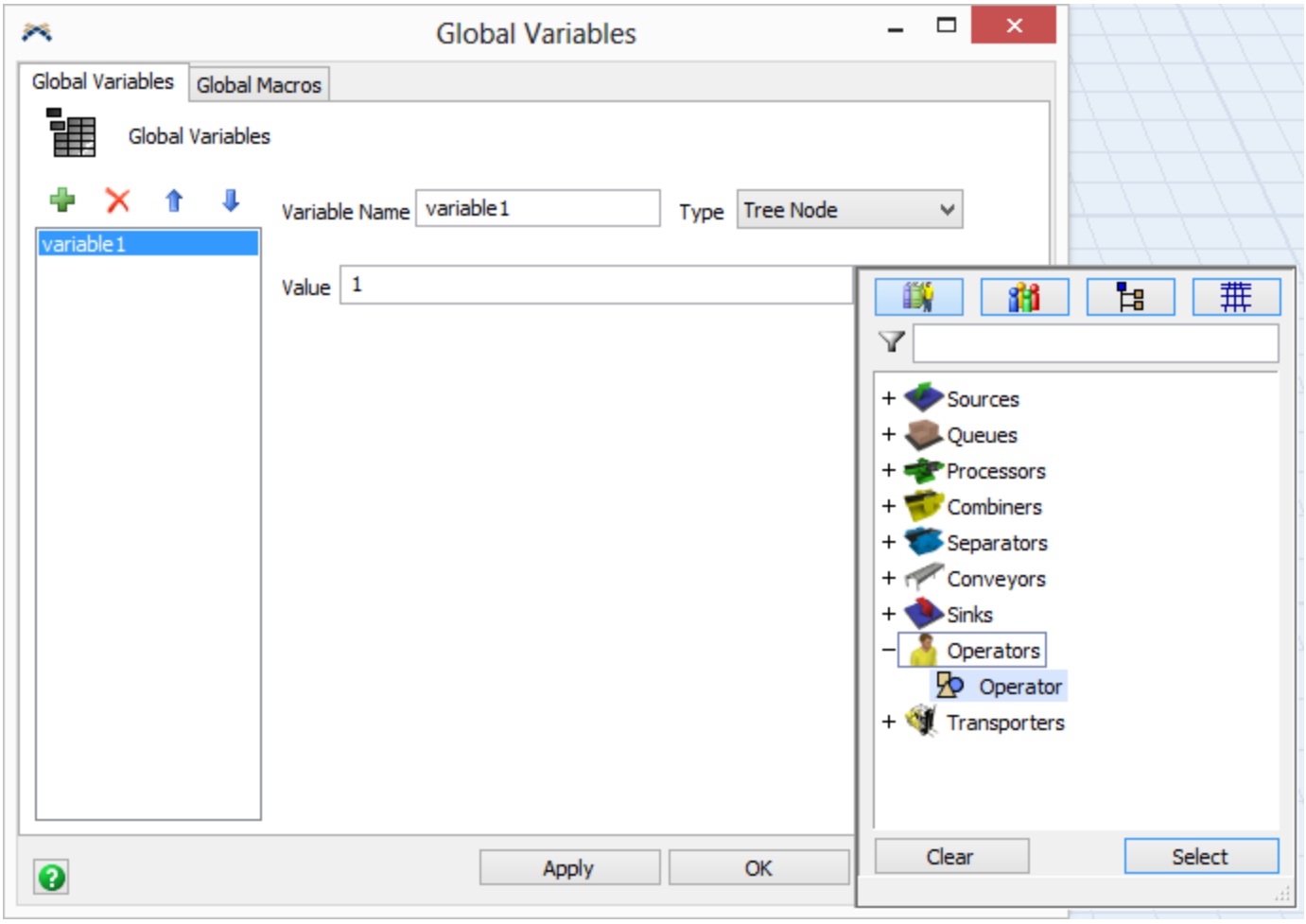
We will also set up a Global Macro to define the processing time of our two processors.

See the Global Variables Window reference to learn more about Global Variables.

- Open the **Global Variables** window from the **Tools** menu.

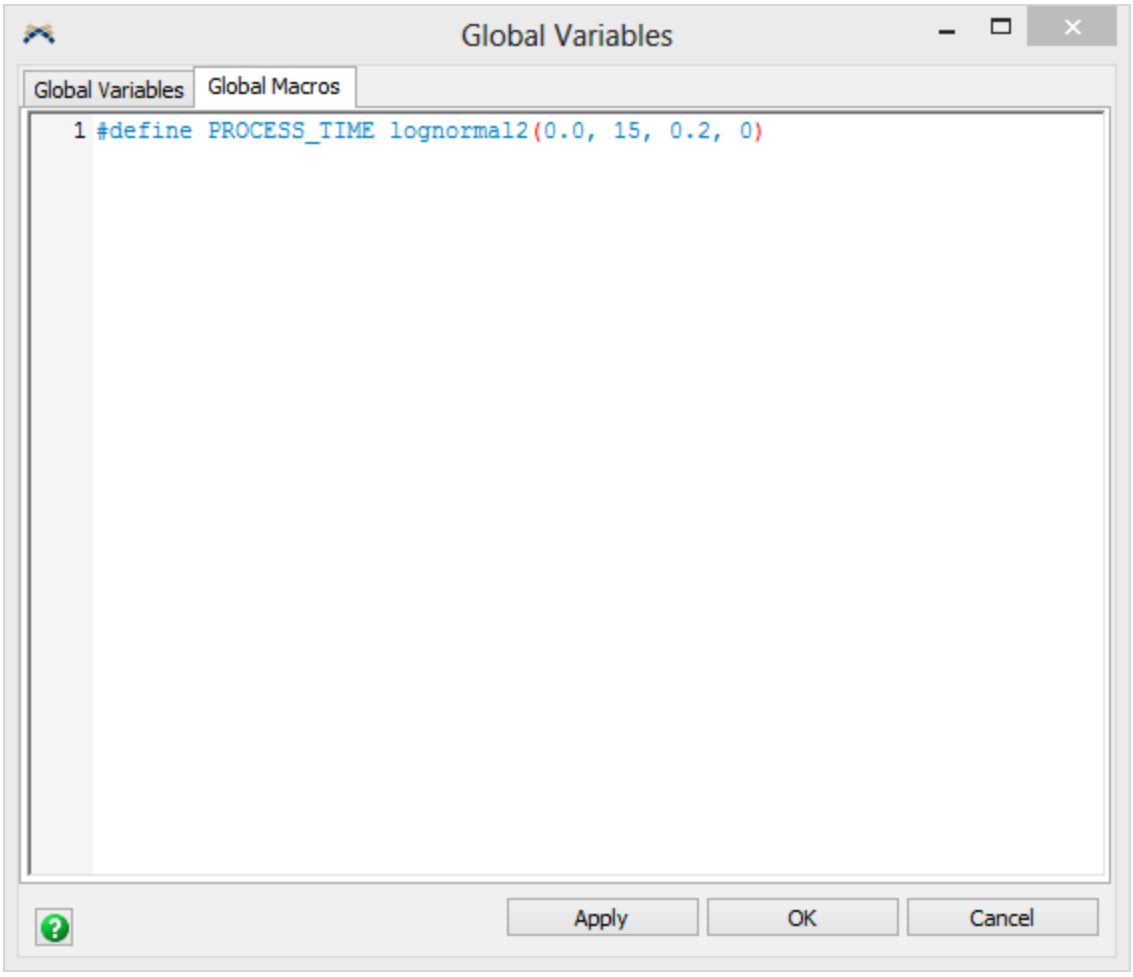


- Click and set the **Variable Name** to *Operator*.
- Select *Tree Node* for the **Type**.
- Click next to the **Value** field and select the *Operator* as the value then click **Select**.



- Repeat the previous steps to create another variable for *Forklift*.
- Next, click on the **Global Macros** tab and enter the follow code:

```
#define PROCESS_TIME lognormal2(0.0, 15, 0.2, 0)
```

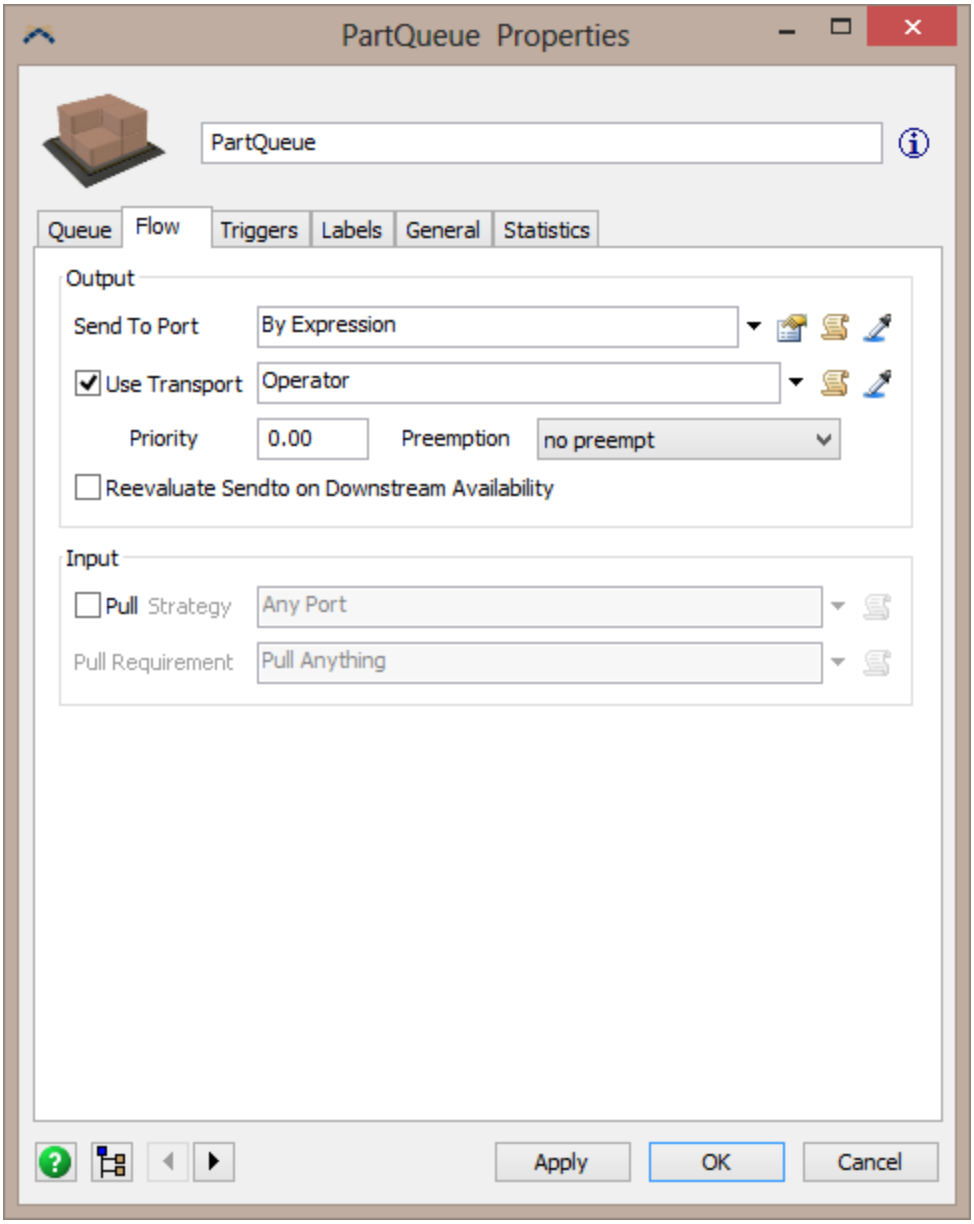


- Click **OK** to apply and close the **Global Variables** window.

Step 6: Setup the Queues

Processor1 is going to handle all items of itemtype 1 and *Processor2* will handle all items of itemtype 2.

- Set the **Maximum Content** of *PartQueue* to 25.
- Under the **Flow** tab, select **By Expression** in the pick list for **Send To Port**. This will default to *getitemtype(item)*.
- Check the **Use Transport** button and replace *centerobject(current, 1)* with *Operator*.

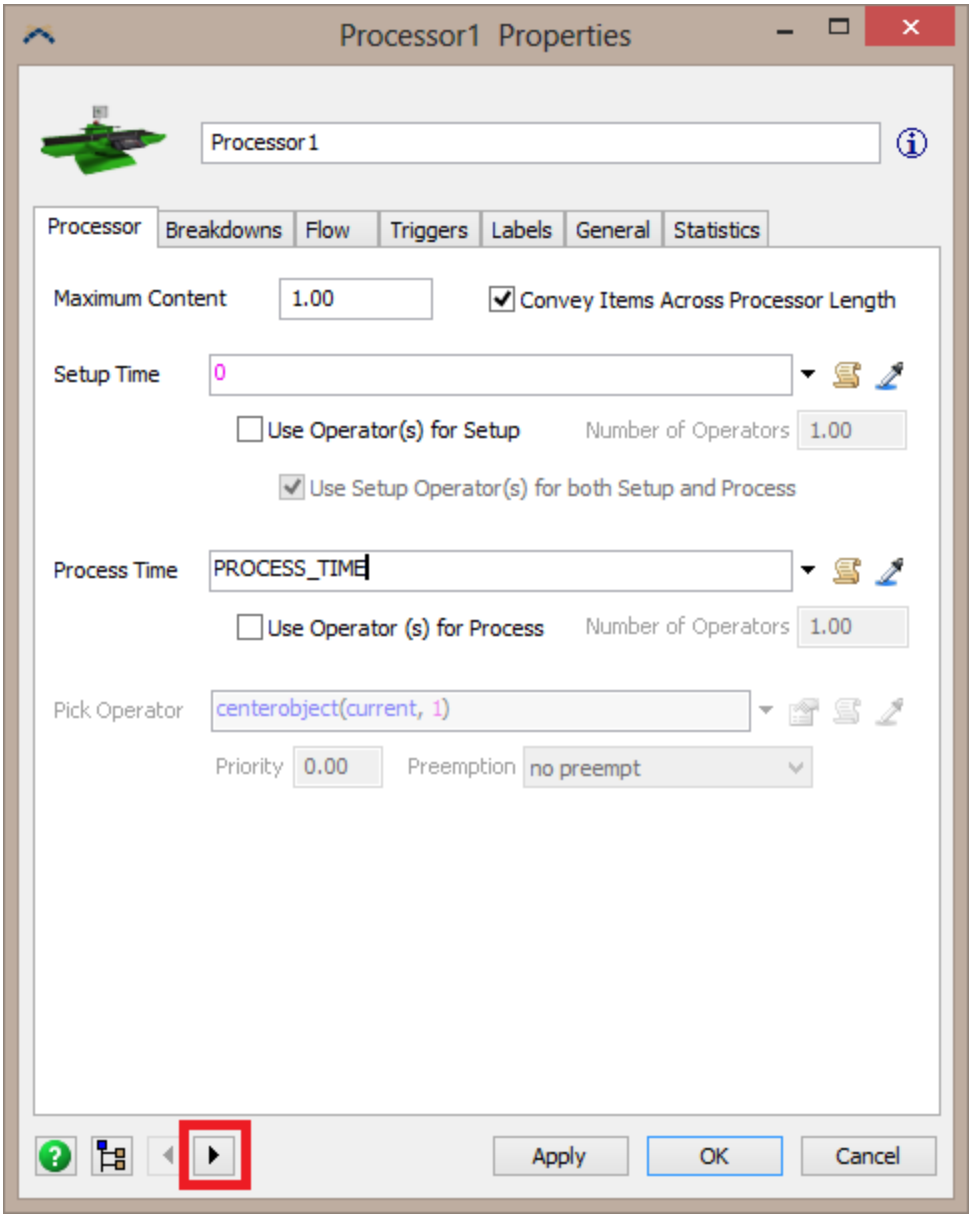


- Click **OK** to apply and close the Properties window.
- Set the **Maximum Content** of *Queue1* and *Queue2* to 100.

Step 7: Setup the Processors

We just defined a Global Macro that can be used for the process time of our processors.

- Open *Processor1* and type PROCESS_TIME into the **Process Time** box.



- Click the **Right Arrow** button at the bottom of the properties window to move to the next processor.
- Set the **Process Time** of *Processor2* to PROCESS_TIME.
- Click **OK** to apply and close the Properties window.

Step 8: Setup a Global Table

We will now set up a Global Table for use with our combiner.

The Combiner is used to group multiple flowitems together. The process by which this works is as follows:

- The Combiner will first accept a single flowitem through input port 1. This becomes the container flowitem into which all the other flowitems will be combined with.
- Once the first flowitem is accepted, the combiner collects a batch of flowitems from the remaining input parts based on its component list.

Components List

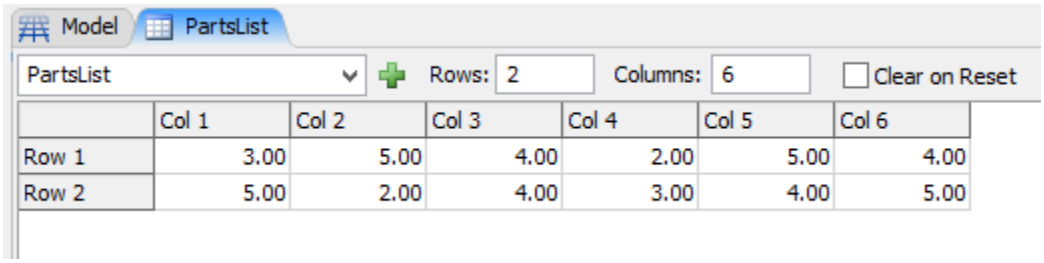
	Target Quantity
From Input Port 2	2.00
From Input Port 3	3.00

- Once the batch is complete, the Combiner goes through its setup and process time before sending the combined flowitems on to the next step of the model.

For this model we're going to have different component lists to represent different orders or assemblies that are being created. Since we connected the pallet to input port 1 it can be used to determine which component list is to be used.

We can create a Global Table that will store all of the different orders (component lists) for use with the Combiner.

- Create a new **Global Table** by going to the **Tools** > **Global Table** menu and select **Add**.
- Rename your table to *PartsList*.
- Set the number of rows to 2 and the number of columns to 6.
- Enter the following numbers into your table.



	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
Row 1	3.00	5.00	4.00	2.00	5.00	4.00
Row 2	5.00	2.00	4.00	3.00	4.00	5.00

- Close the Global Table window.

Step 9: Setup the Combiner

- Open the *Combiner* properties window and click on the **Combiner** tab.
- Select **Convey Items Across Combiner Length**.
- Click on the **Flow** tab and select **Use Transport**.
- Replace *centerobject(current,1)* with *Forklift*.

We want the Combiner's Component List to update each time a new pallet comes into port 1.

- Click on the **Triggers** tab and add a new operation to the **OnEntry** trigger.
- Select the *Update Combiner Component List* from the pick list.
- Replace *"tablename"* with *"PartsList"*.
- Click **OK** to apply and close the Properties window.

The *Update Combiner Component List* pulls data from a Global Table based on the itemtype of the incoming flowitem on port 1. So, if the pallet is of itemtype 4, it will pull the two row values from column 4 of our Global Table *PartsList*.

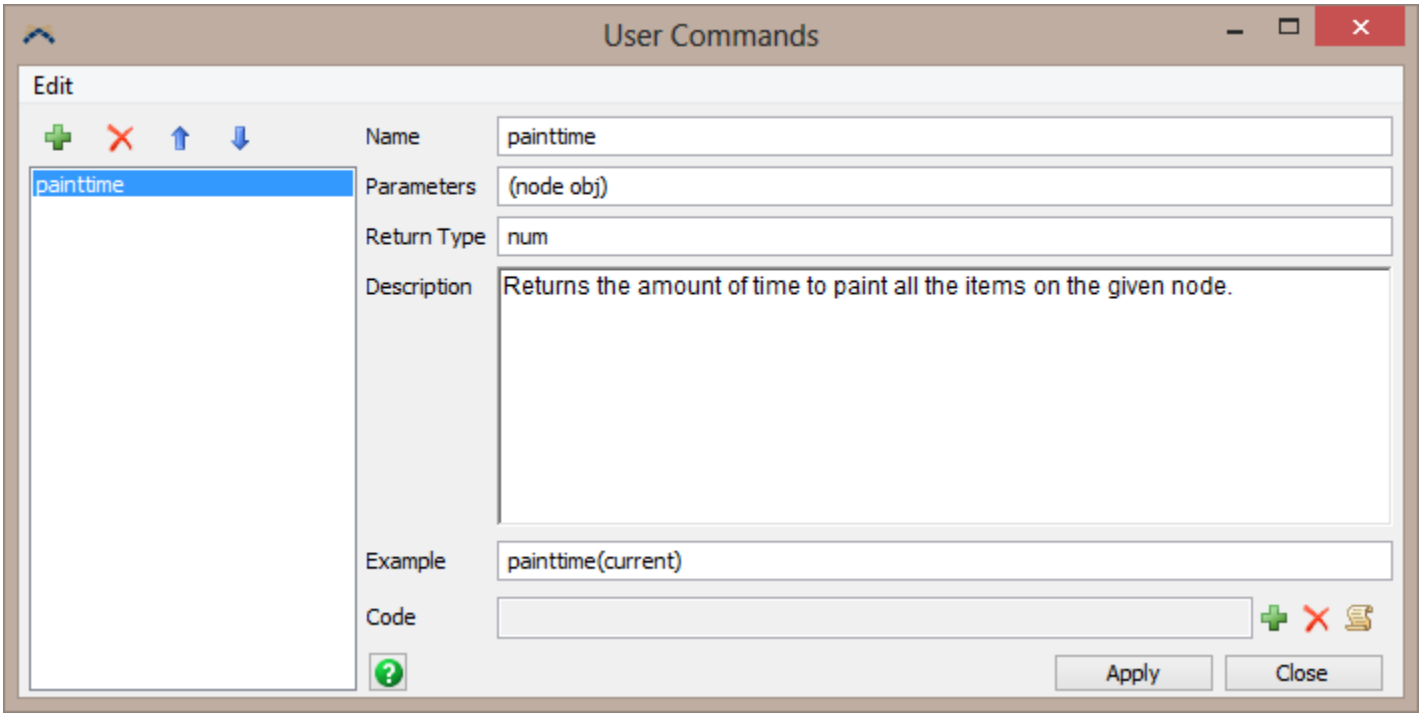
Step 10: Write Custom User Commands

FlexSim allows you to write your own custom User Commands, or functions, that can be used throughout your model.

The first command we will write will return the time it takes to paint all of the items on our pallet. This code could be written directly into the process time of the *Painter1* and *Painter2* processors, but as we have multiple painters all utilizing the same processing time, writing a custom command makes it easier to make changes to that processing time.

- Open the **User Commands** window by going to **Tools** > **User Commands**.
- Click **Add** to create a new command.
- Set the **Name** of the command to *painttime*.

- Set the **Parameters** to *(node obj)*.
- Set the **Return Type** to *num*.
- The command's **Description** is: *Returns the amount of time to paint all the items on the given node.*
- Set the **Example** to *painttime(current)*.



For the code, we want our command to pass the *Painter* processor, cycle through all of the items on the pallet, and return a total time for the painting where items with itemtype 1 take 20 seconds to paint and items of itemtype 2 take 14 seconds to paint.

- Click on the button to open the code for our command.
- Enter the following code:

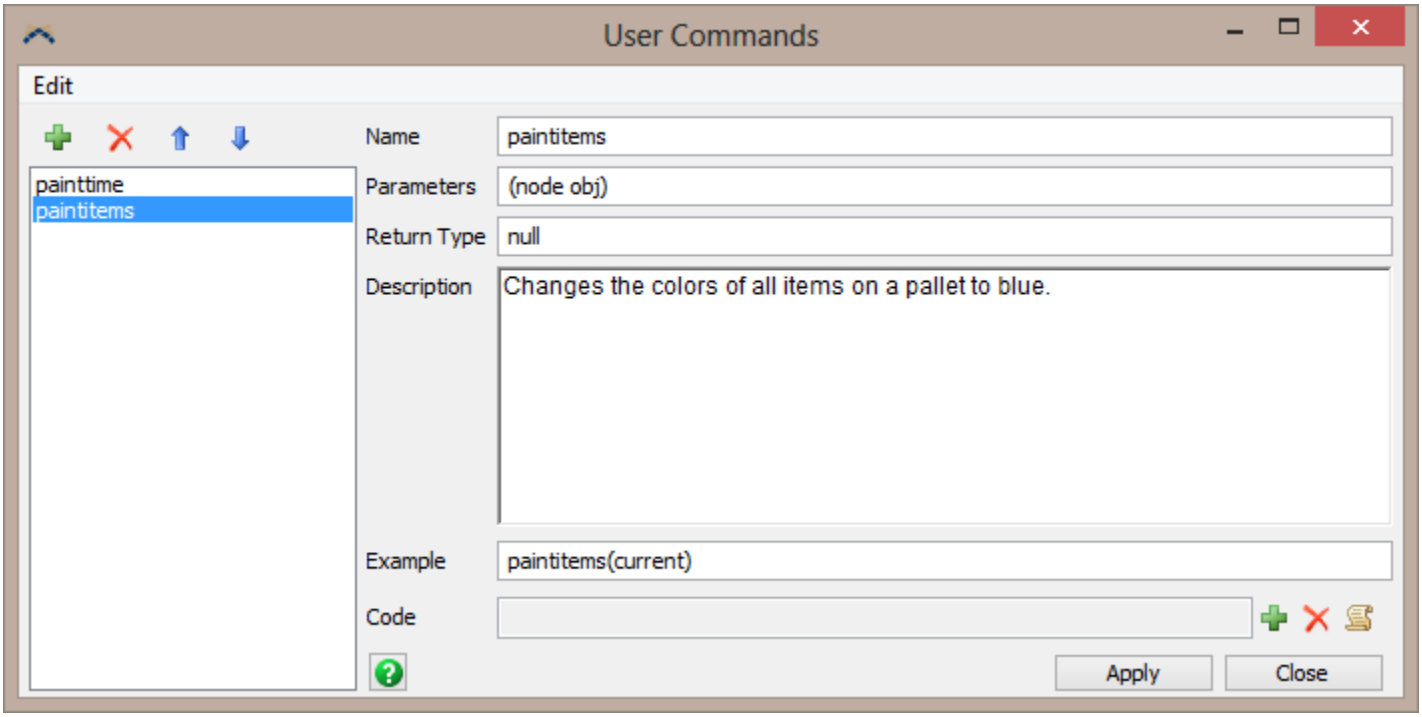
```
treenode object = parnode(1);
int painttime = 0;
for(int index = 1; index < content(first(object)); index++) {
     if(getitemtype(rank(object,index)) == 1)      {
             painttime += exponential(0.0, 20.0, 0);
     } else {
             painttime += exponential(0.0, 14.0, 0);
     }
}
return painttime;
```

- Click the **OK** button to apply and close the code window.

Next we will create a command that will change the color of all items on the pallet to be blue once they've gone through the painter.

- Add a new user command and set the **Name** to *paintitems*.
- Set the **Parameters** to *(node obj)*.

- Set the **Return Type** to *null*.
- The command's **Description** is: *Changes the colors of all items on a pallet to blue.*
- Set the **Example** to *paintitems(current)*.



For the code, we will cycle through all of the items on the pallet and change the color to blue.

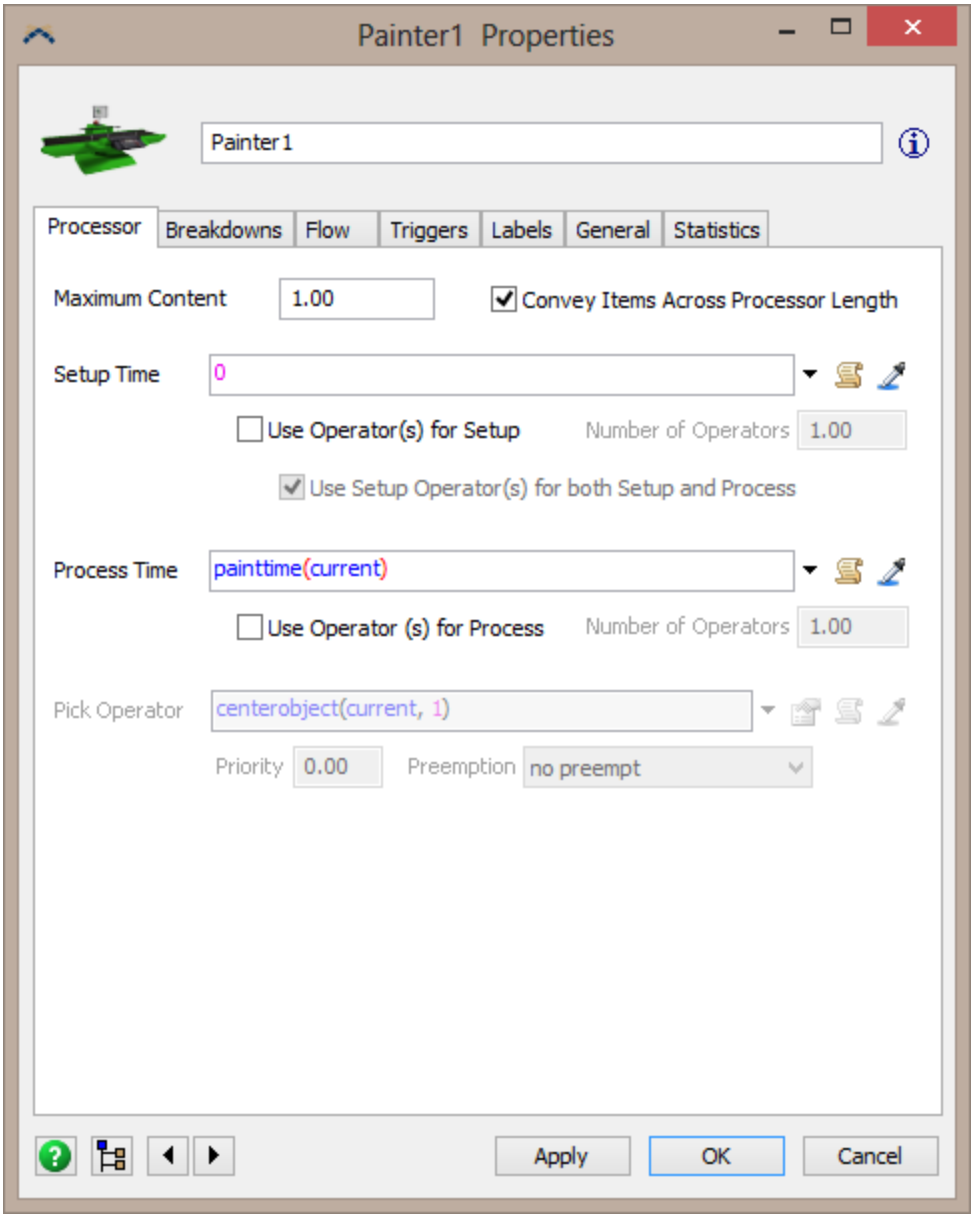
- Click on the button to open the code for our command.
- Enter the following code:

```
treenode object = parnode(1);
for(int index = 1; index <= content(first(object)); index++) {
     colorblue(rank(first(object),index));
}
```

- Click the **OK** button to apply and close the code window.
- Click **Apply** and then **Close** on the **User Commands** window.

Step 11: Setup the Painters

- Open *Painter1's* properties window.
- Set the **Process Time** to *painttime(current)*.



- Click the **Flow** tab and check the **Use Transport** button.
- Set the transport to be *Forklift*.
- Click on the **Triggers** tab and click on the next to the **OnProcessFinish** trigger to edit the code of that trigger.
- Add the following line to the bottom of the code:

```
paintitems(current);
```

- Click **OK** to apply and close to the code window.
- Click **OK** to apply and close the Properties window.

Set up the same properties for *Painter2*. For a fast way to copy properties from one object to another, see the Edit Selected Objects Utility.

Step 12: The Separator

As the *Separator* currently stands, we do not need to make any changes to its properties. By default, when the *Separator* receives a flowitem, it separates the container from the contents and then sends the

container (pallet) through output port 1 and the contents through output port 2. If you wish to change these settings, go to the **Flow** tab of the *Separator* and change the **Send To Port** option.

We are now ready to run the model.

Step 13: Reset and Run the Model

- Remember to hit the

button to reset model parameters to their initial state.
- Click the

button to start the simulation.

You should see flowitems entering the queue and then being moved by the Operator to one of the two processors. Red items will move to *Processor1* and green items will move to *Processor2*.

Pallets will be place on the *Combiner* and then wait until it receives its full batch of items from *Queue1* and *Queue2*. The *Forklift* will then move the pallet to *Painter1* or *Painter2*. Before exiting the painters, all of the items will turn blue. The pallet will then be moved to the *Separator* where the pallet will be separated from the items. The pallet will travel across *Conveyor1* and the blue items will travel across *Conveyor2* to the *Sink*.

User Events Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

This tutorial will help you understand how to use User Events in your simulation. It assumes you know how to use Global Variables. If you don't, please see our Global Modeling Tools Tutorial.

What You Will Learn

- How to use the User Events Tool.

Approximate Time to Complete this Lesson

This lesson should take about 15-20 minutes to complete.

Model Overview:

In this model we will relocate operators every time we reset the model, as well as set the inter-arrival rate and number of processors to use. At a given time in the model we will change the rate of part arrivals and open up additional processors.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

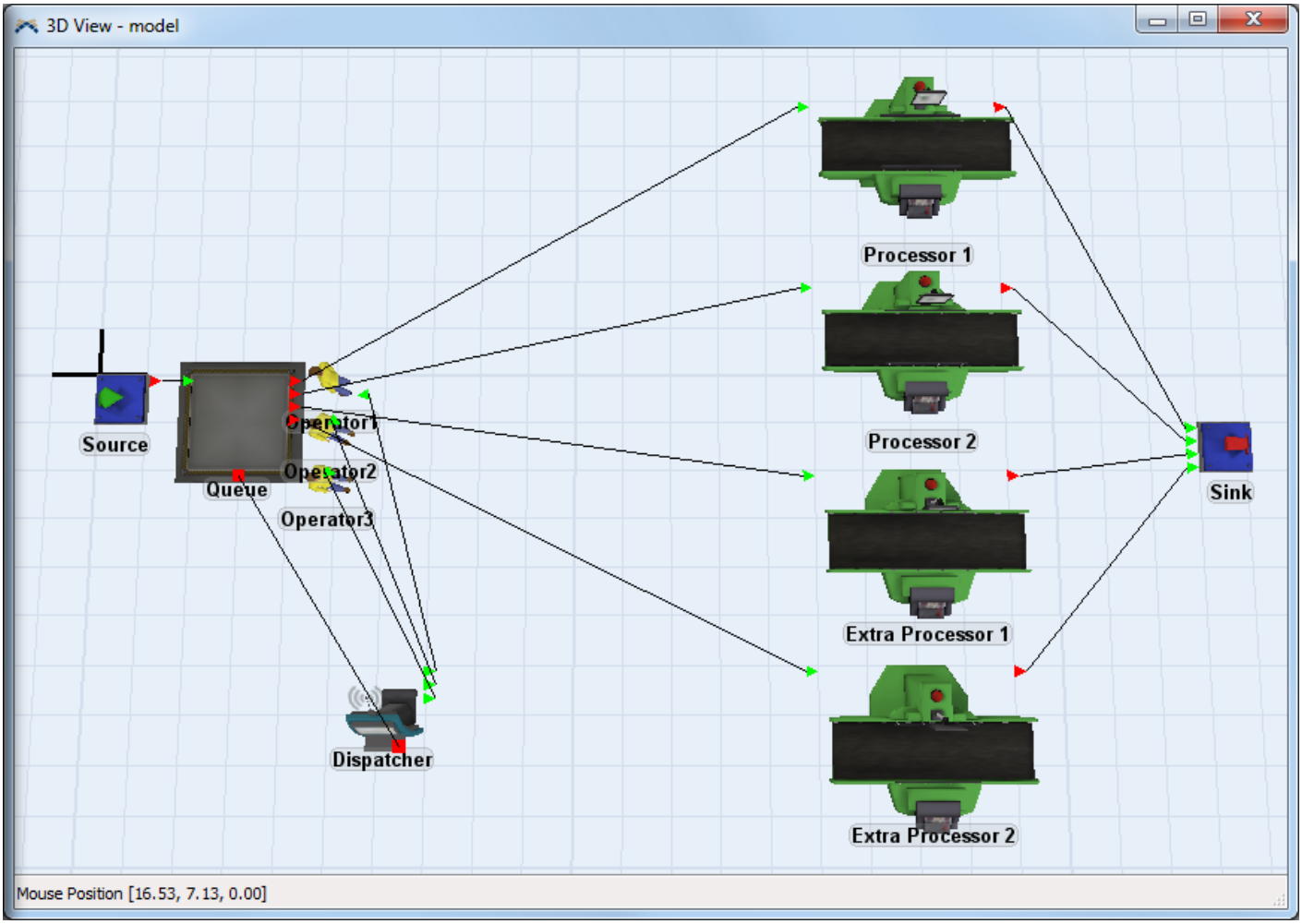
Building Labels Model

Begin a new model by clicking the button on the toolbar. Click OK on the Model Units window, we will use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.
- Rename the objects as shown.



Connect all of the objects as shown:

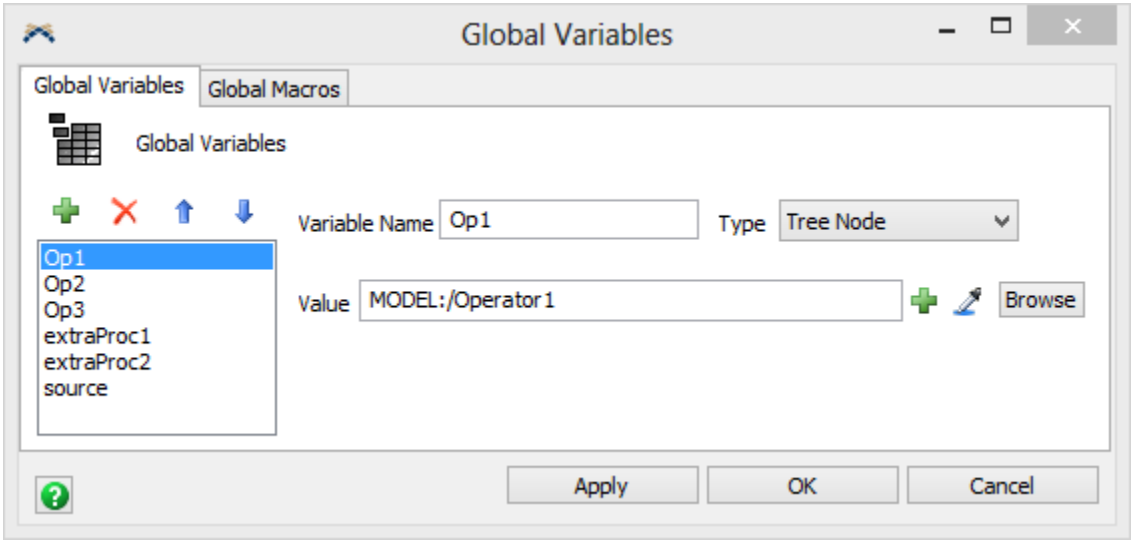
- Connect *Source* to *Queue*.
- Connect *Queue* to *Processor 1*, *Processor 2*, *Extra Processor 1* and *Extra Processor 2*.
- Connect *Processor 1*, *Processor 2*, *Extra Processor 1* and *Extra Processor 2* to *Sink*.
- Connect *Queue* to *Dispatcher* with a centerport connection (S key).
- Connect *Dispatcher* to *Operator1*, *Operator2* and *Operator3* with a standard connection (A key).

Step 2: Setup Global Variables

Here we will create global variables for objects that we will access in our User Event. Alternatively, objects can be accessed by the node command:

```
treenode Op1 = node("/Operator1", model());
```

- Open the **Global Variables** window (Tools > Global Variables).
- Create a global variable for each operator, the two extra processors, and the source. Each of them will be of type *Tree Node*, named as shown:



Step 3: Setup Source and Queue

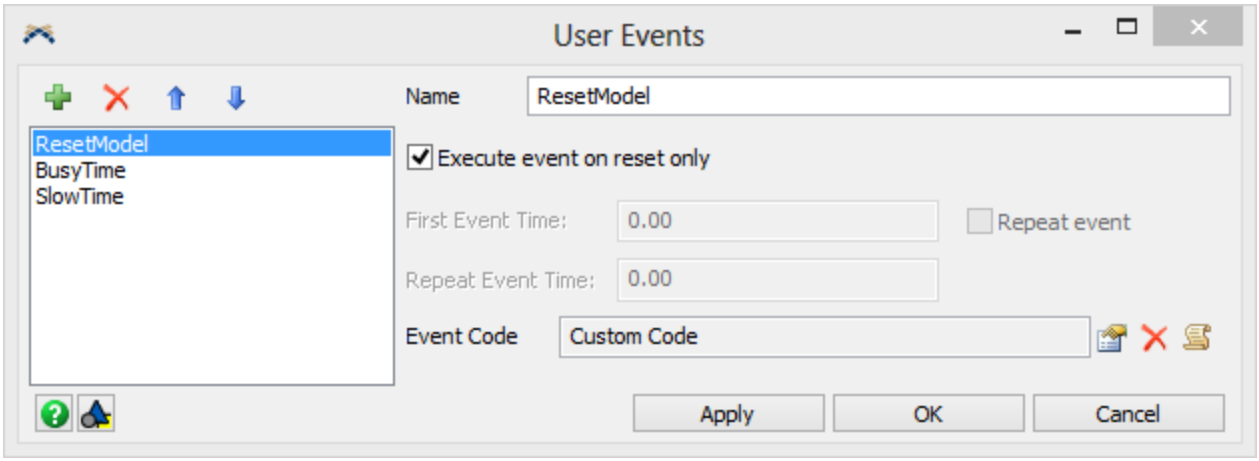
- Open the *Source* Properties Window.
- Go to the **Labels** tab.
- Create a number label called *"arrivalTime"*, leave it set to 0.
- Go to the *Source* tab.
- For the **Inter-arrival time** enter *getlabelnum(current, "arrivalTime")*.
- Click Ok to apply and close the properties window.

- Open the *Queue* properties window.
- Go to the **Flow** tab.
- Check **Use Transport**.

Step 4: Create User Events

We will create three user events. One to fire when the model resets and two to change the "busyness" of the model.

- Open the **User Events** window (Tools > User Events).
- Add three new events and name them *"ResetModel"*, *"BusyTime"* and *"SlowTime"*.



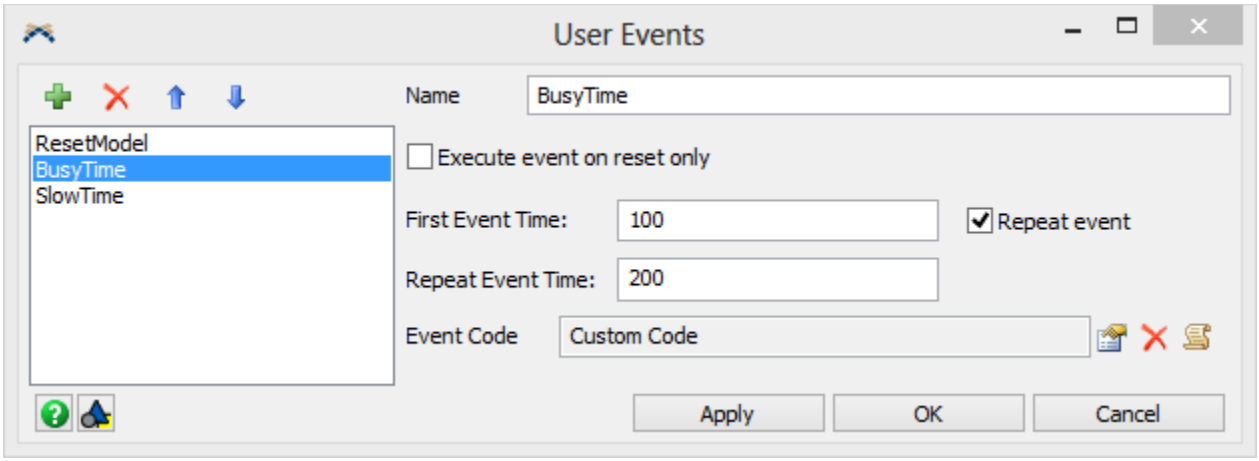
- With *ResetModel* selected, check **Execute event on reset only**. Every time you press the Reset button, this code will be executed.
- Click the code edit button and write in the following code:

```
setloc(Op1, 5, 0, 0);
setloc(Op2, 5, -1, 0);
setloc(Op3, 5, -2, 0);
setlabelnum(source, "arrivalTime", 10);
```

```
1 /**Custom Code*/
2 treenode current = ownerobject(c);
3
4 setloc(Op1, 5, 0, 0);
5 setloc(Op2, 5, -1, 0);
6 setloc(Op3, 5, -2, 0);
7
8 setlabelnum(source, "arrivalTime", 10);
```

This will set the location of each operator next to the queue (if your queue isn't in the same place as mine [x:2.00, y:0.00, z:0.00], check the location of your own queue and enter appropriate x, y and z values). This user event will also set the label on the source to 10, so the Inter-arrival time will change accordingly.

- With *BusyTime* selected, enter *100* for **First Event Time** and *200* for **Repeat Event Time**. Every 200 seconds the model will go into busy time, starting at time 100.



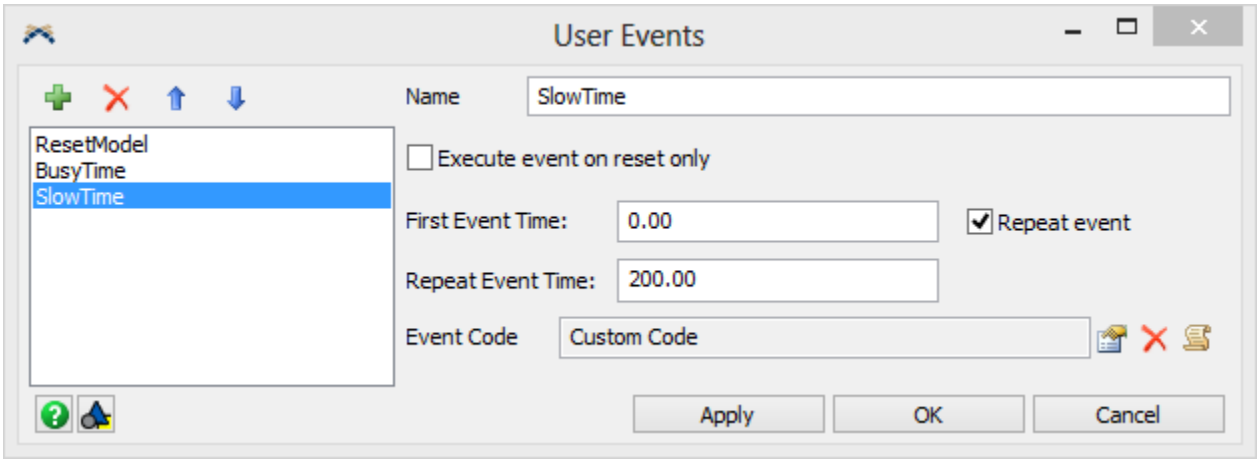
Click the code edit button and write in the following code:

```
setlabelnum(source, "arrivalTime", 5);
openinput(extraProc1);
openinput(extraProc2);
msg("Busy Time", "It's Busy Time!");
```

```
1 /**Custom Code*/
2 treenode current = ownerobject(c);
3
4 setlabelnum(source, "arrivalTime", 5);
5
6 openinput(extraProc1);
7 openinput(extraProc2);
8
9 msg("Busy Time", "It's Busy Time!");
```

This will set the inter-arrival time to 5, so parts are created twice as fast. The extra processors will open so they can be used during busy time. You will also get a message telling you it is busy time.

- With *SlowTime* selected, enter *0* for **First Event Time** and *200* for **Repeat Event Time**. Every 200 seconds the model will go into slow time, starting at time 0.



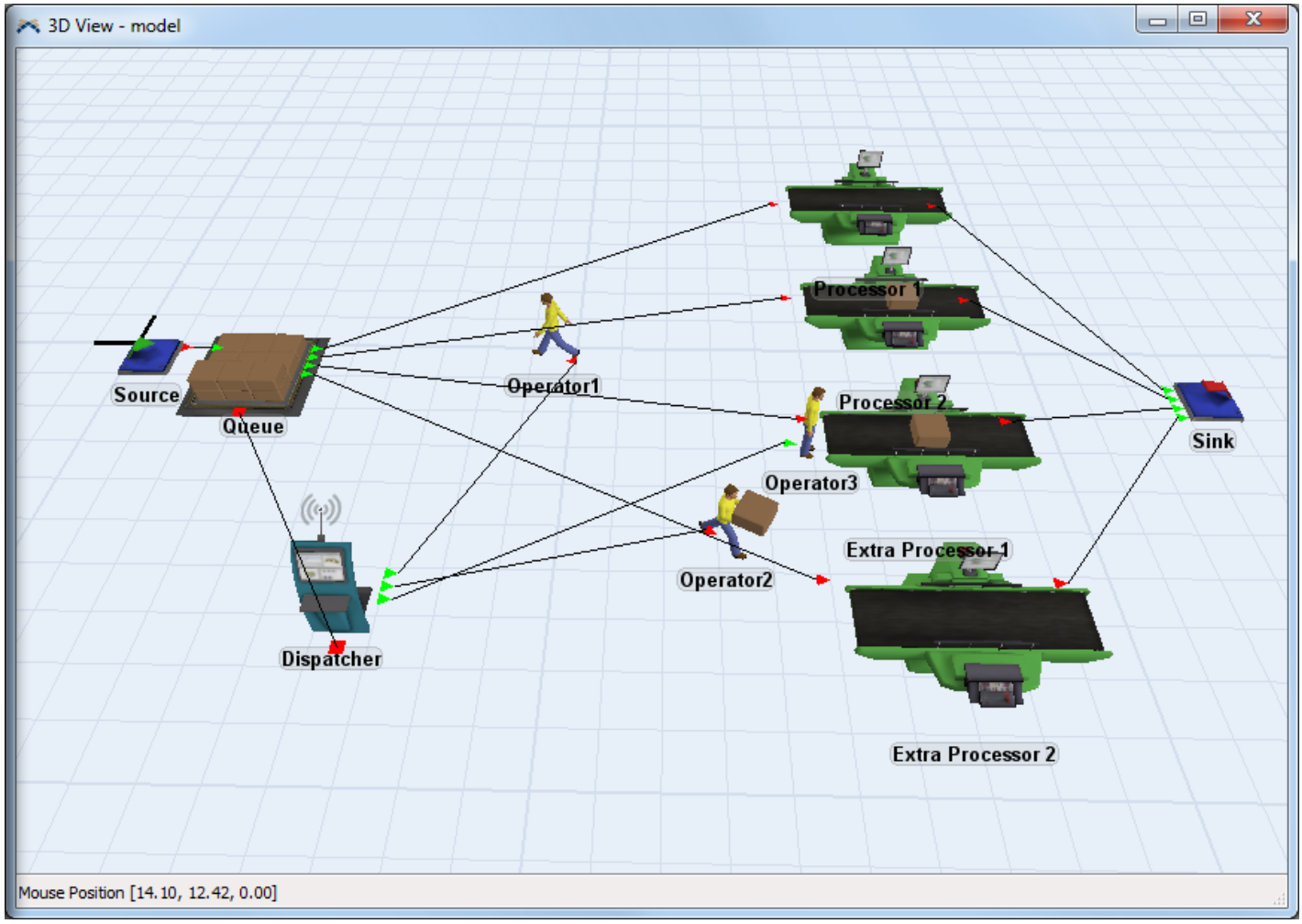
- Click the code edit button and write in the following code:

```
closeinput(extraProc1);
closeinput(extraProc2);
setlabelnum(source, "arrivalTime", 10);
```

```
1 /**Custom Code*/
2 treenode current = ownerobject(c);
3
4 closeinput(extraProc1);
5 closeinput(extraProc2);
6
7 setlabelnum(source, "arrivalTime", 10);
```

This will set the inter-arrival time to 10, so parts come slow again. The extra processors will closed during slow time.

- Press OK on the **User Events window** to close and apply your changes.
- Reset and run your model. Notice that the operators are always reset to the same place and it switches between busy and slow times.



This completes the User Events tutorial. Congratulations!

TimeTables Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

This tutorial will introduce you to the TimeTable tool. The TimeTable can be used to specify specific times when a FixedResource or a TaskExecuter are scheduled to be down. This could be due to a break, maintenance, repairs, etc.

What You Will Learn

- How to create TimeTables and assign members.
- How to use a TimeTable to specify down times for your processors and operators.

Approximate Time to Complete this Lesson

This lesson should take about 15-20 minutes to complete.

Model Overview

In this model we will have multiple operators performing a task. A TimeTable will be created to specify when an operator is on break. Another TimeTable will schedule maintenance for the processors.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

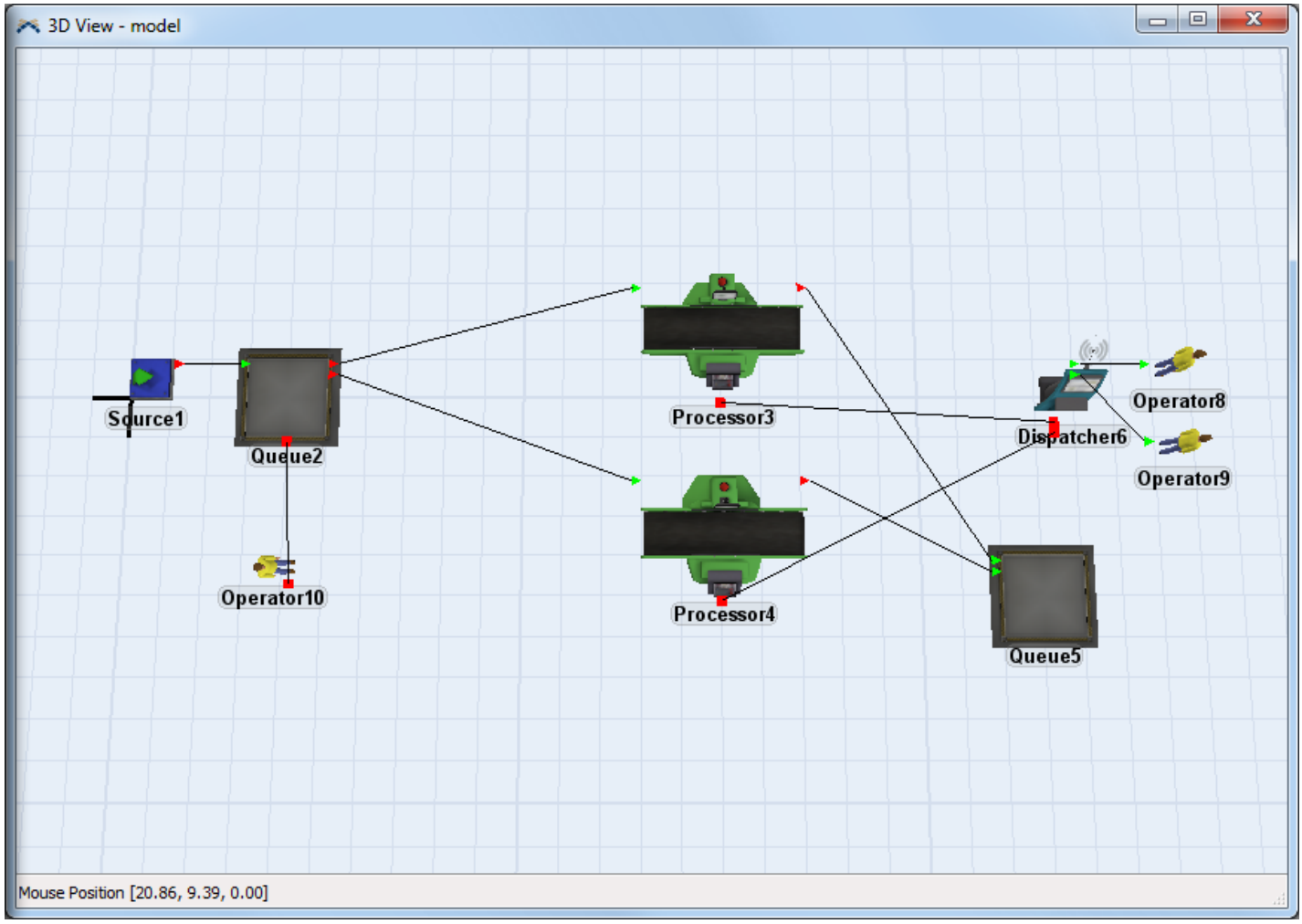
Building TimeTable Model

Begin a new model by clicking the button on the toolbar. Click OK on the Model Units window, we will use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.



Connect all of the objects as shown:

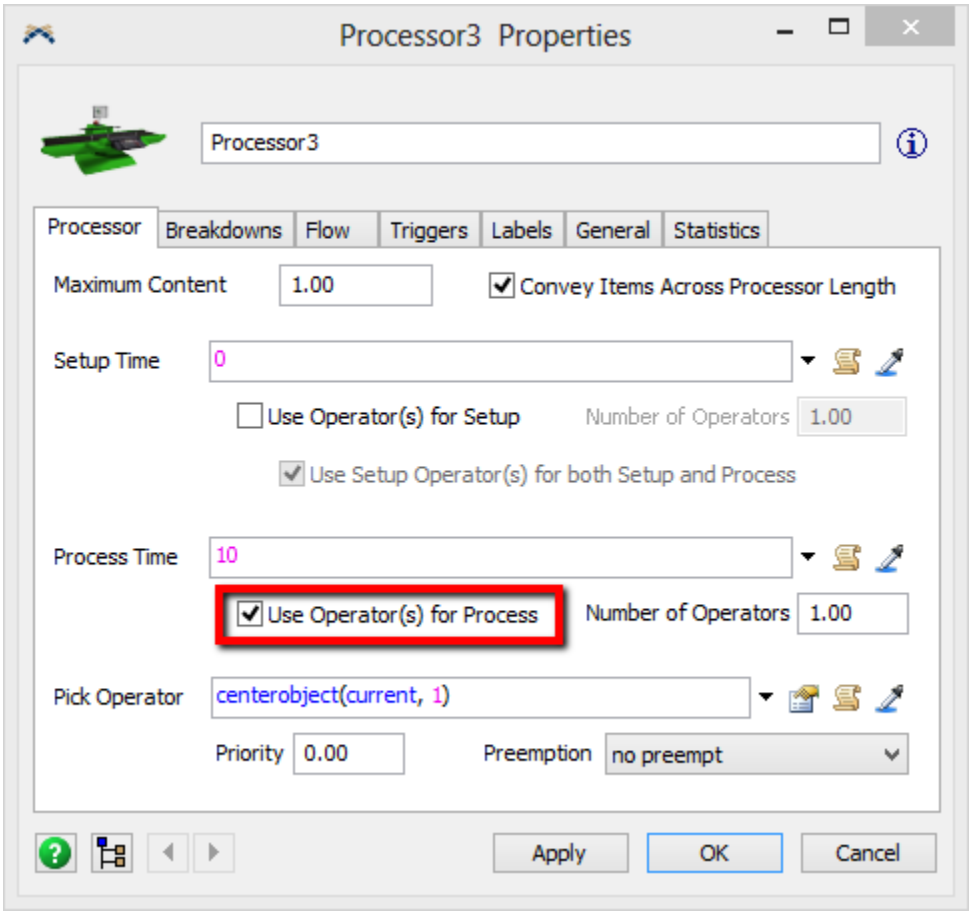
- Connect *Source1* to *Queue2*.
- Connect *Queue2* to *Processor3* and *Processor4*.
- Connect *Processor3* and *Processor4* to *Queue5*.
- Connect *Dispatcher6* to *Operator8* and *Operator9*.
- Connect *Queue2* to *Operator10* with a centerport connection (S key).
- Connect *Processor3* and *Processor4* to *Dispatcher6* with a centerport connection (S key).

Step 2: Setup the Queue and Processors

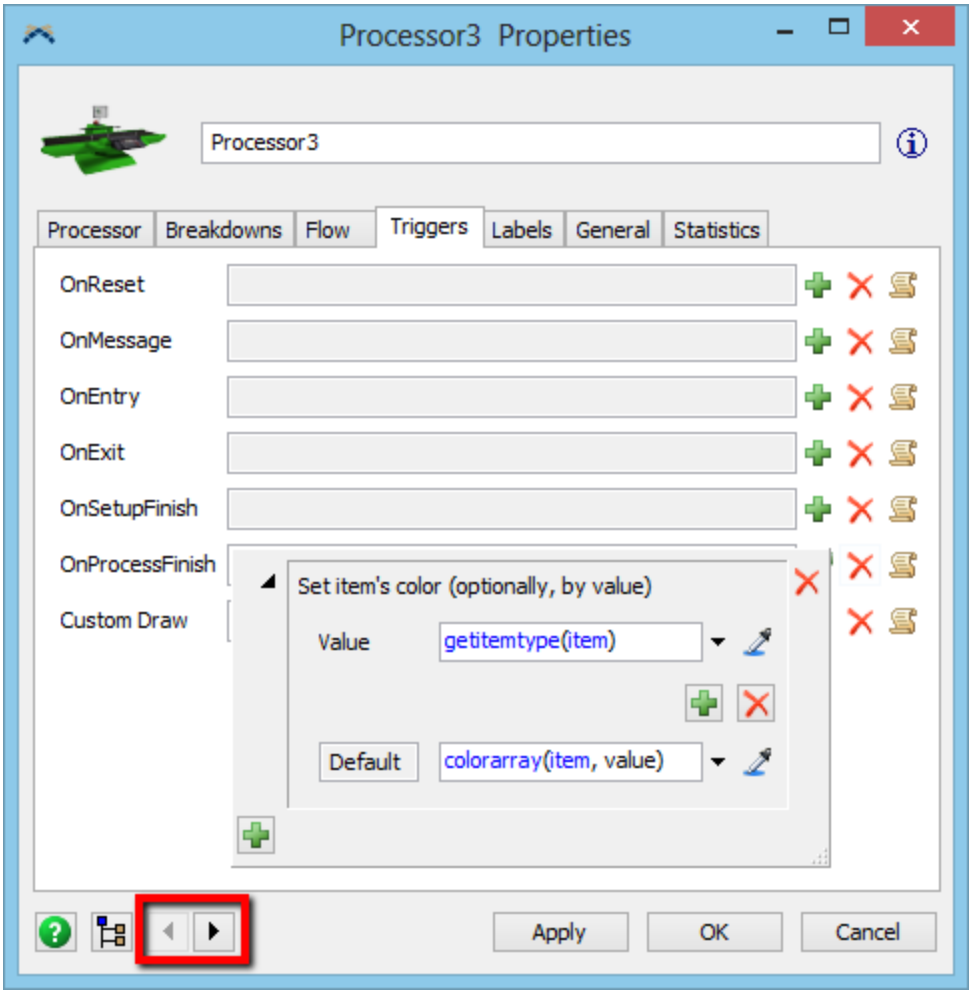
One operator will transport flowitems from *Queue2* to the two processors. The other two operators will be used to process flowitems at the two processors and to transport flowitems from the processors to *Queue5*.

- Click on *Queue2* to open its properties in the Quick Properties window.
- Under the **Flow** section, check **Use Transport** and leave it using the default centerobject option.

- Open the properties window of *Processor3*.
- On the **Processor** tab, check **Use Operator(s) for Process** and leave it using the default centerobject option.



- Go to the **Flow** tab.
- Check **Use Transport** and leave it using the default centerobject option.
- Go to the **Triggers** tab.
- Click the add button for the **OnProcessFinish** trigger.
- Select the *Set Color* option and leave it on the default options.



Note: You can switch quickly between processors (or other similar objects) by clicking on the left and right arrow buttons at the bottom of any properties window.

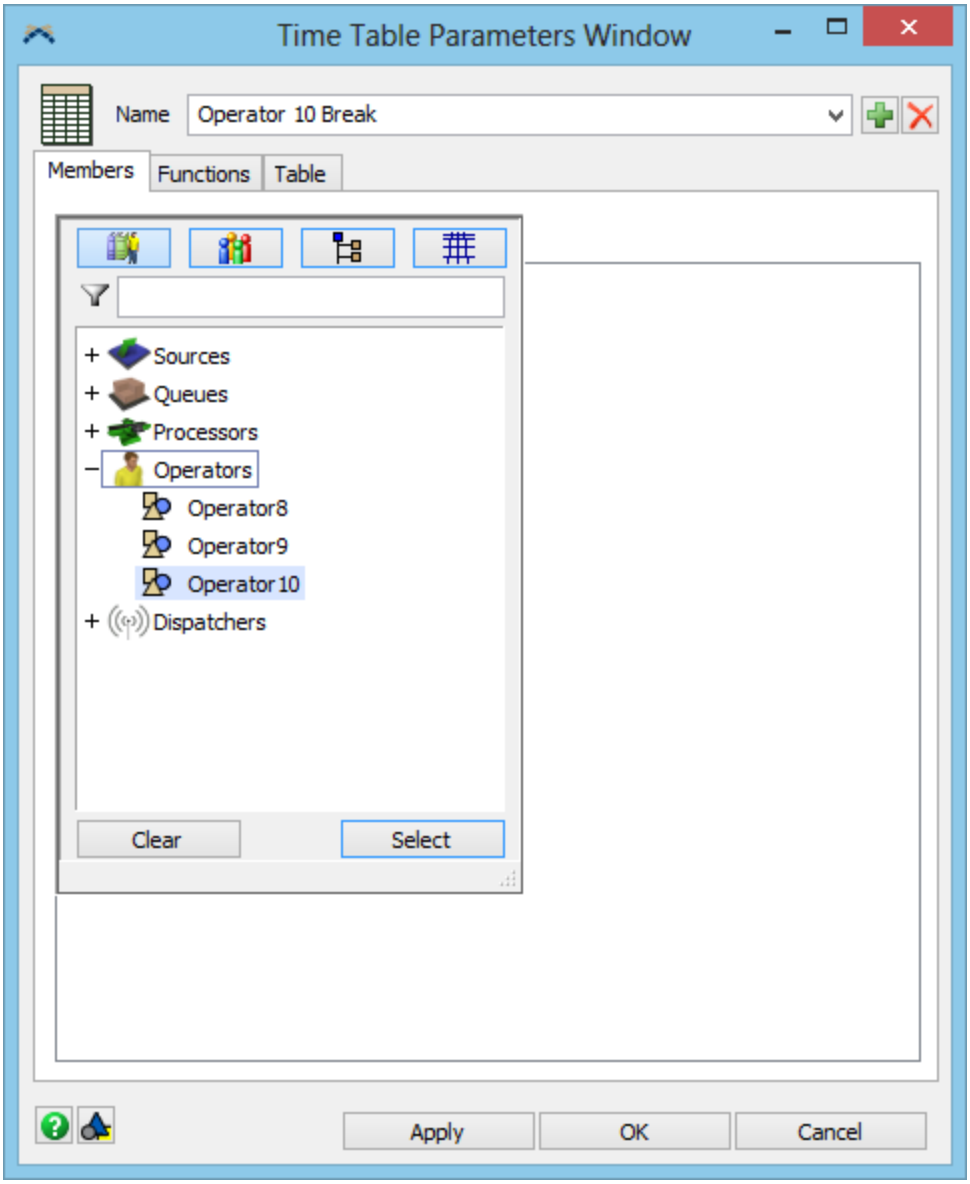
- Repeat the last set of steps for *Processor4*.
- Click **OK** to apply and close the properties window.

Reset and run the model to make sure the operators are taking the boxes from *Queue2* to the processors, processing the boxes and then taking them to *Queue5*. The boxes should also be changing color after processing.

Step 3: Create a TimeTable

We will now create a TimeTable for *Operator10*.

- Open a **TimeTable** window by going to **Tools > Time Tables** and click **Add**.
- Rename the **TimeTable** to *Operator 10 Break*.
- Go the Members tab and click the button to add a member. Highlight *Operator10* and click **Select**.
- Go to the Table tab and in row 1 of the table, set **Time** to *200*, set **State** to *12*, and set **Duration** to *30*.
- Set the **Repeat** to Custom and change the value to *200*. This will cause the operator to go on break every 200 seconds.
- Go to the Functions tab and in the pick list for the **Down Function**, select *Travel to Location, Delay Until Down Time Complete*. Change only the **Location** to *2, -8, 0*. These are the x, y and z coordinates, respectively, that the operator will go to while on break.



- In the pick list for the **Resume Function**, select *Do Nothing*.
- Click **OK** to apply and close the TimeTable window.

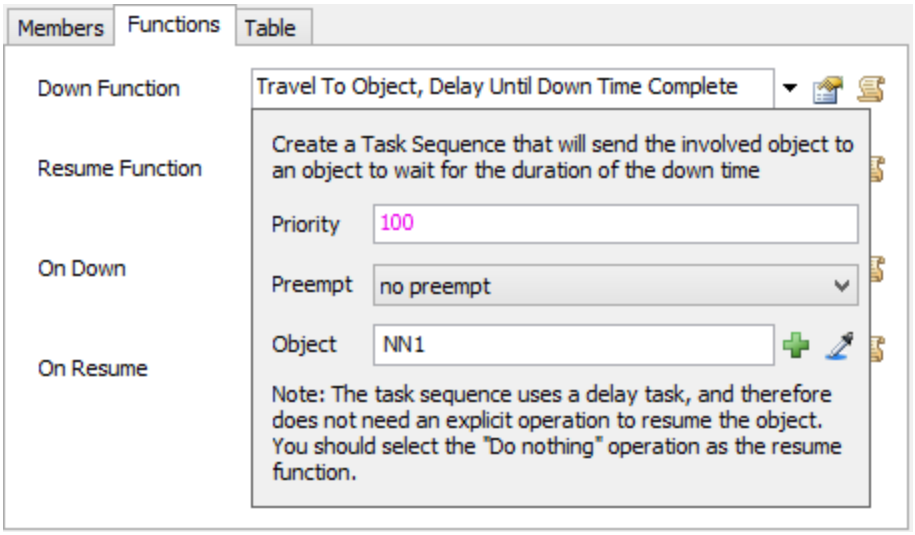
Reset and Run your model. When you run you model, at 200 seconds you will see that the operator walks away from the work area for 30 seconds and then returns to work.

Note: If the operator is in the middle of a task, he will complete the task before starting his down time.

Step 4: Update the TimeTable

We can also have the operator break to a specific object instead of a set of coordinates.

- Create a *Network Node* object and place it away from your work area.
- Go back to your *Operator 10 Break* TimeTable.
- In the pick list for the **Down Function**, select *Travel to Object, Delay Until Down Time Complete*. Change the **Destination Name** to *NN1*.


Members
Functions
Table
Down Function
Travel To Object, Delay Until Down Time Complete
Resume Function
Create a Task Sequence that will send the involved object to an object to wait for the duration of the down time
Priority
100
On Down
Preempt
no preempt
Object
NN1
On Resume
Note: The task sequence uses a delay task, and therefore does not need an explicit operation to resume the object. You should select the "Do nothing" operation as the resume function.

- Leave the pick list for the **Resume Function** on *Do Nothing*.
- Click **OK** to apply and close the TimeTable window.
- Reset and Run your model and notice that the operator will go to *NN1*.

WARNING: The duration of your operator's break will be the time he spends at that location. If he takes 10 seconds to walk there, he will still break for 30 seconds once he gets there. Adding in his 10 second journey back, he will total 50 seconds taken between finishing his last task and starting his new one.

Step 5: Schedule Maintenance for the Processor

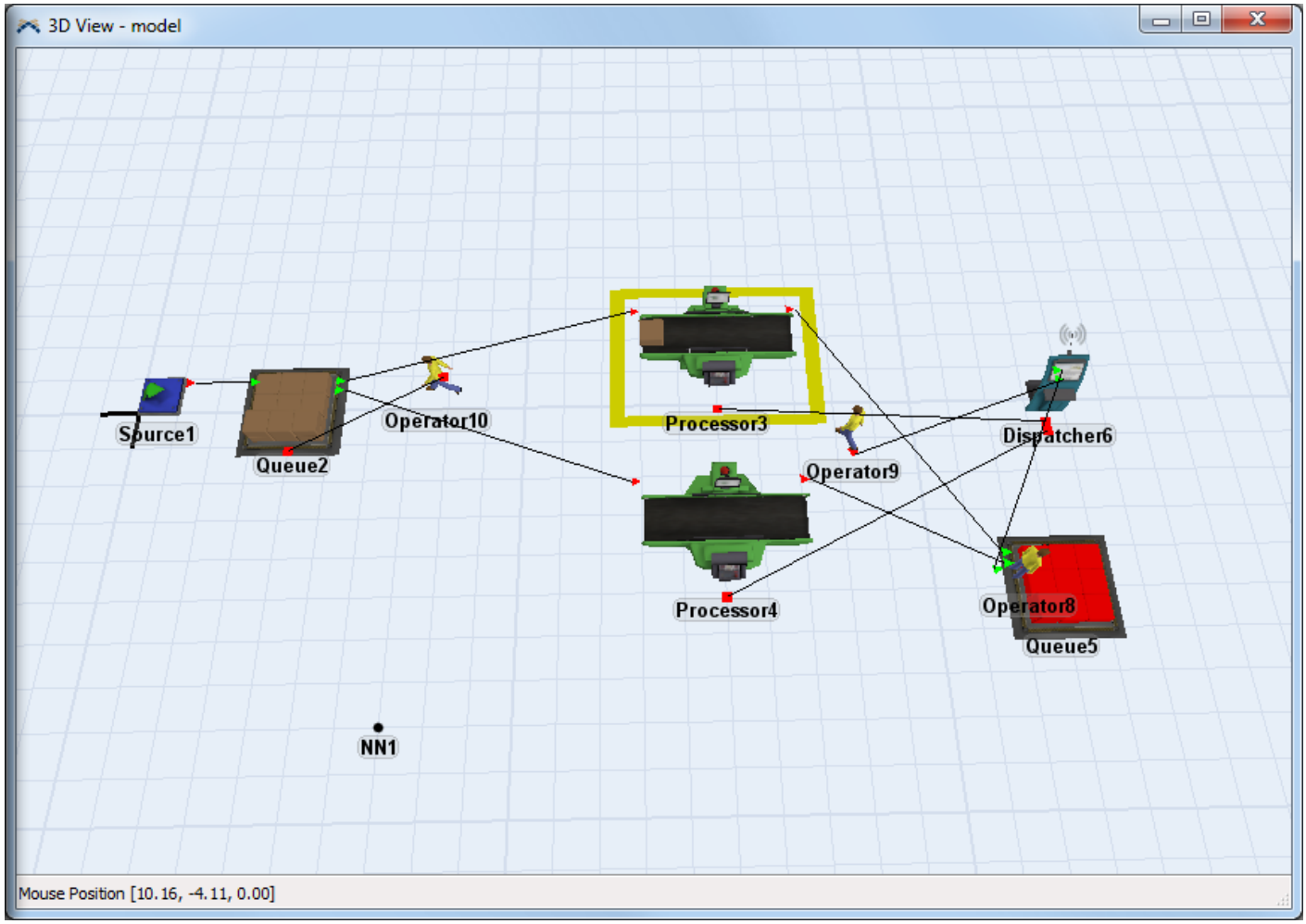
Next, we will give **Processor3** a scheduled downtime for maintenance.

- Add another **TimeTable**.
- Rename the TimeTable to *Processor Down Time*.
- Add the member *Processor3*.
- Under the Table tab, in Row 1 of the table, set **Time** to *200*, set **State** to *12* and set **Duration** to *100*.
- Set the **Repeat Time** to Custom, *300*. This will cause the processor to go down every 300 seconds after the initial maintenance.
- In the pick list for the **Down Function**, select *Stop Input*.
- In the pick list for the **Resume Function**, select *Resume Input*.

Stopping and resuming the object's input will mean that it will continue to process any item that is currently in it, but will not receive anymore items until the down time is done. If you leave the TimeTable at the default Stop/Resume Object, the processor will stop with or without an item in it and not resume until the down time is complete.

- Click **OK** to apply and close the TimeTable window.

Reset and run your model. Your model should look something like this:



This completes the TimeTables tutorial. Congratulations!

Kinematics Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

This tutorial will introduce you to using Kinematics in your model. Kinematics allow you to perform simultaneous movements with a single object. This is best seen by looking at the way the Crane object moves.

What You Will Learn

- How to create and add Kinematics to a fixed resource object.
- How to move an object using Kinematics.

Approximate Time to Complete this Lesson

This lesson should take about 30-45 minutes to complete.

Model Overview

In this model we will treat a processor as if it is a centrifuge and have it spin as it processes flowitems.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

Building Kinematics Model

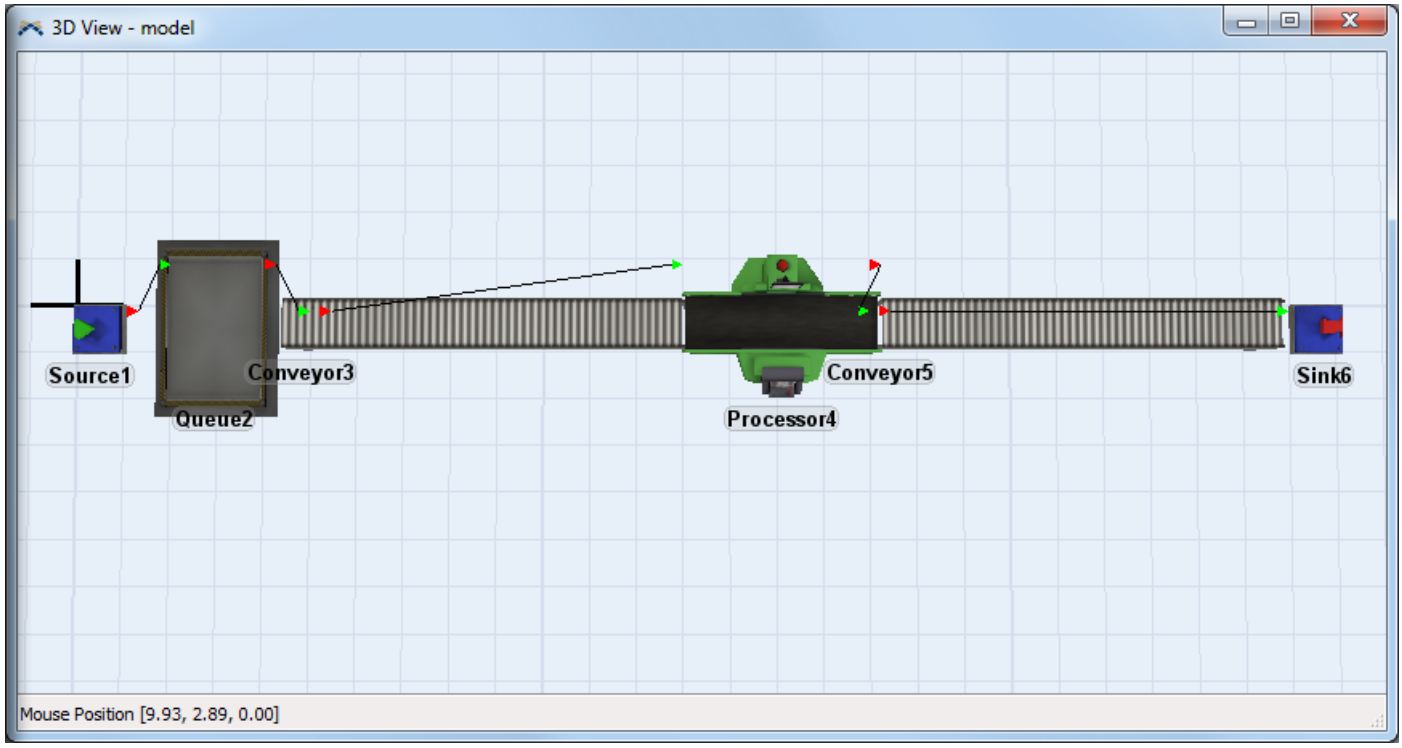
Begin a new model by clicking the button on the toolbar. Click **OK** on the Model Units window, we will use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Part 1: Basic Kinematics

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.



Connect all of the objects as shown:

- Connect *Source1* to *Queue2* to *Conveyor4* to *Processor3* to *Conveyor5* to *Sink6*

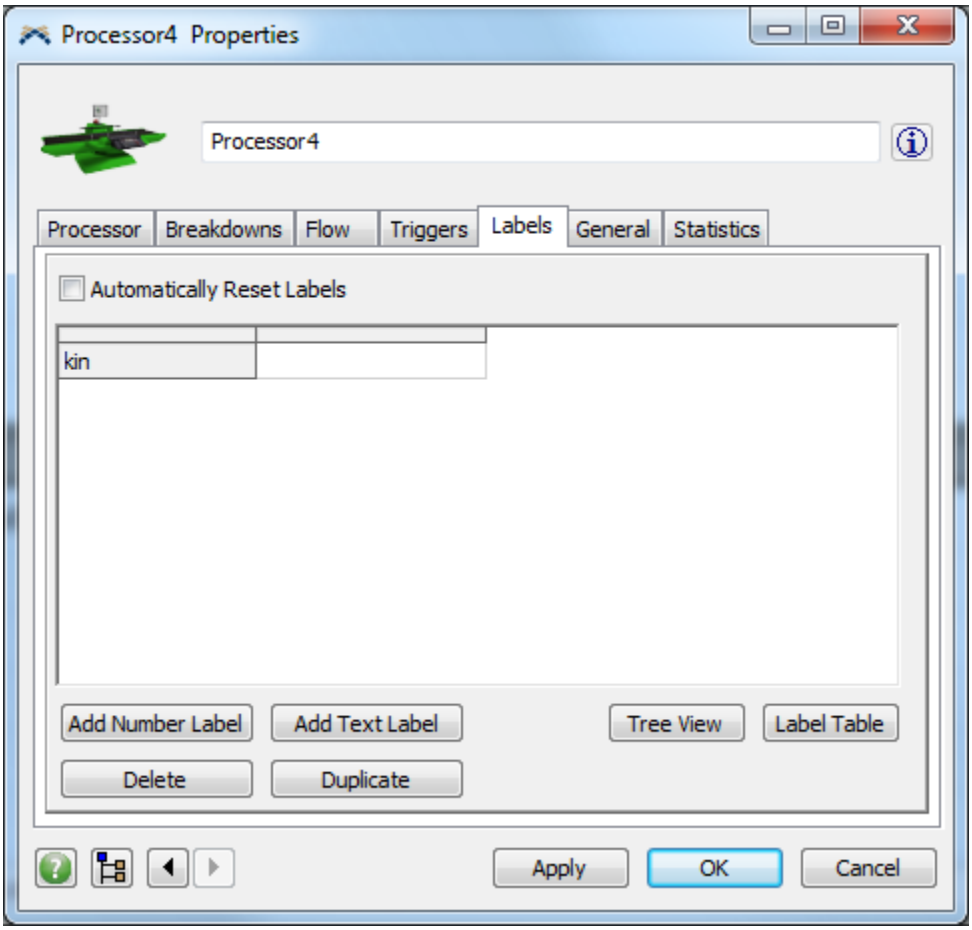
Step 2: Add Kinematics Label

All kinematics need a unique node for the information to be stored. The simplest way to do this is by creating a text label dedicated to the kinematics.

Alternatively, a node could be added to the object's variables, in which case of all the *label* commands would be replaced with *getvarnode*.

- Open the properties window of the *Processor*.
- Go to the **Labels** tab.
- Add a text label by clicking the **Add Text Label** and rename the label *"kin"*.

WARNING: Do not try to delete the kinematics node from within the properties window. Use the following instructions to change the parameters of your kinematic. Deleting it manually may crash the program.

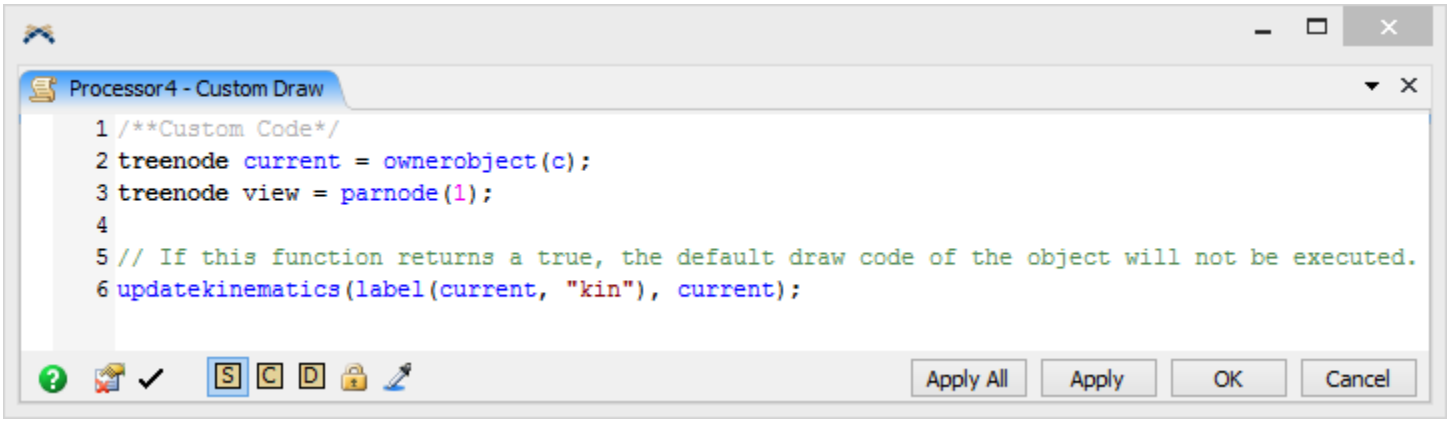


Step 3: Add Custom Draw Code

The kinematics will need to be updated continually as the model runs.

- Go to the **Triggers** tab.
- Click the code edit button next to the **Custom Draw** trigger.
- Enter the following line of code:

```
updatekinematics(label(current, "kin"), current);
```


Processor4 - Custom Draw
1 /**Custom Code*/
2 treenode current = ownerobject(c);
3 treenode view = parnode(1);
4
5 // If this function returns a true, the default draw code of the object will not be executed.
6 updatekinematics(label(current, "kin"), current);
Apply All
Apply
OK
Cancel

- Click **OK** to apply and close the code edit window.

Step 4: Update Kinematics when Process Finishes

The kinematics will need a final update at the time they should be complete. This will ensure that, regardless of the framerate of your 3D view (which defines how often the Custom Draw trigger fires), or whether you have any 3D views open at all, the kinematic will still be properly updated to its final resting position/rotation.

- In the **Triggers** Tab, click the code edit button next to the **OnProcessFinish** trigger.
- Copy the same *updatekinematics* command used in the **Custom Draw** trigger:

```
updatekinematics(label(current, "kin"), current);
```

```
1 /**Custom Code*/
2 treenode  item = parnode(1);
3 treenode  current = ownerobject(c);
4
5 updatekinematics(label(current, "kin"), current);
```

- Click **OK** to apply and close the code edit window.

Step 5: Add OnReset Code

You want your object to return to its original position when you reset the model.

- In the **Triggers** tab, click the code edit button next to the **OnReset** trigger.
- Use the *initkinematics* command with the node being the text label you just created. Switch over to the **General** tab of the properties window and check the x, y, and z position and rotation of the object. Set the corresponding values in the *initkinematics* command. (The rotation values of your processor will be 0 by default.)
- Set the last two parameters to *0*.

```
initkinematics(label(current, "kin"), x, y, z, 0, 0, 0, 0, 0);
```

Note: The last two parameters will indicate rotation management and local coordinates. When rotation management is set to 1, your object will rotate so the the "front" (the positive x-direction) of it is facing the direction of travel. If local coordinates is set to 1, it will use the coordinates of the object's container instead of the model itself.

```
1 /**Custom Code*/
2 treenode current = ownerobject(c);
3
4 initkinematics(label(current, "kin"), 12, 1, 0,0,0,0,0,0);
```

- Click **OK** to apply and close the code edit window.

Step 6: Update Kinematics when Setup Finishes

You will also want your object to return to its original state at the start of the kinematic. We will now add the kinematic information that will move the object.

- In the **Triggers** tab, click the code edit button next to the **OnSetupFinish** trigger.
- Copy the *initkinematics* command with the same parameters as in the **OnReset** trigger.
- Enter the following line of code:

```
addkinematic(label(current, "kin"), 0, 0, 3240, 360, 90, 180, 0, 0, time(), 2);
```

```
1 /**Custom Code*/
2 treenode  item = parnode(1);
3 treenode  current = ownerobject(c);
4
5 initkinematics(label(current, "kin"), 12, 1, 0,0,0,0,0,0);
6 addkinematic(label(current, "kin"), 0, 0, 3240,360, 90, 180, 0, 0, time(), 2);
```

The *addkinematic* command sets the x, y, and z parameters, to 0, 0, and 3240, respectively. This will be a rotational motion so it will rotate around the z axis 3240 degrees (9 turns). The target speed (or maximum speed) is set to 360 degrees/sec, with an acceleration of 90 degrees/sec/sec and deceleration of 180 degrees/sec/sec. The start speed and end speed are 0. The start time will be the time that the command is called, so we use the time() command. Last of all, we want this to be a rotational motion so we set the parameter to 2 or KINEMATIC_ROTATE (for translational motion we would set this to 1 or KINEMATIC_TRAVEL).

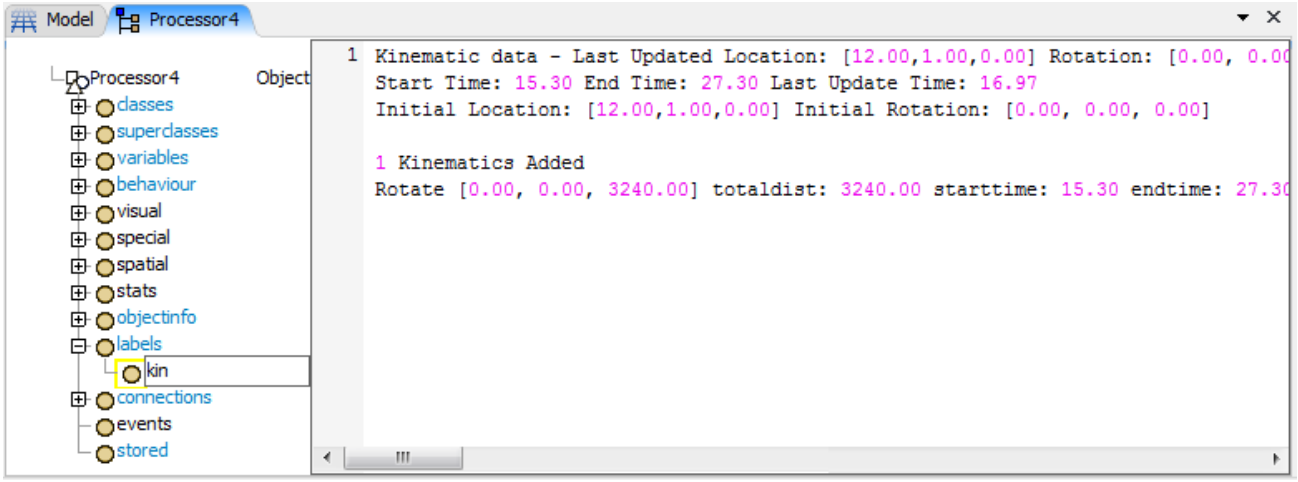
- Click **OK** to apply and close the properties window.

Reset and run the model and you should see the processor accelerate, spin and decelerate to a stop. You may notice that if the next flowitem starts being processed before your kinematic is done that it will instantaneously reset its position to match the *initkinematics* parameters. Our next step will be to match the process time with the time it takes for you kinematic to end.

Step 7: View the Kinematic Information

Important information about your kinematic will be stored in the *"kin"* label.

- Run the model until a flowitem enters the processor and stop the model. DO NOT reset.
- Right click on the *Processor* and click the **Explore Tree** button.
- Expand the processor tree, then expand the *"labels"* node within that tree. Click on the *"kin"* label and you will see the information for the kinematic as in the following figure:



- Scroll to the right until you can see the starttime and endtime. The difference between the two is the time it takes for the kinematic to complete. You should see a difference of 12 seconds.

- Open the *Processor's* properties window.
- Under the **Processor** tab, change the **Process Time** to *12*.
- Optional: Adding a **Setup Time** of a second or two will allow our item to move out of the processor before the processor begins spinning again.

When you run the model now you should see that the processor finishes processing when it is done spinning. Kinematics lets you set up your simulation in terms of speeds and accelerations of your equipment as well as giving you the visual.

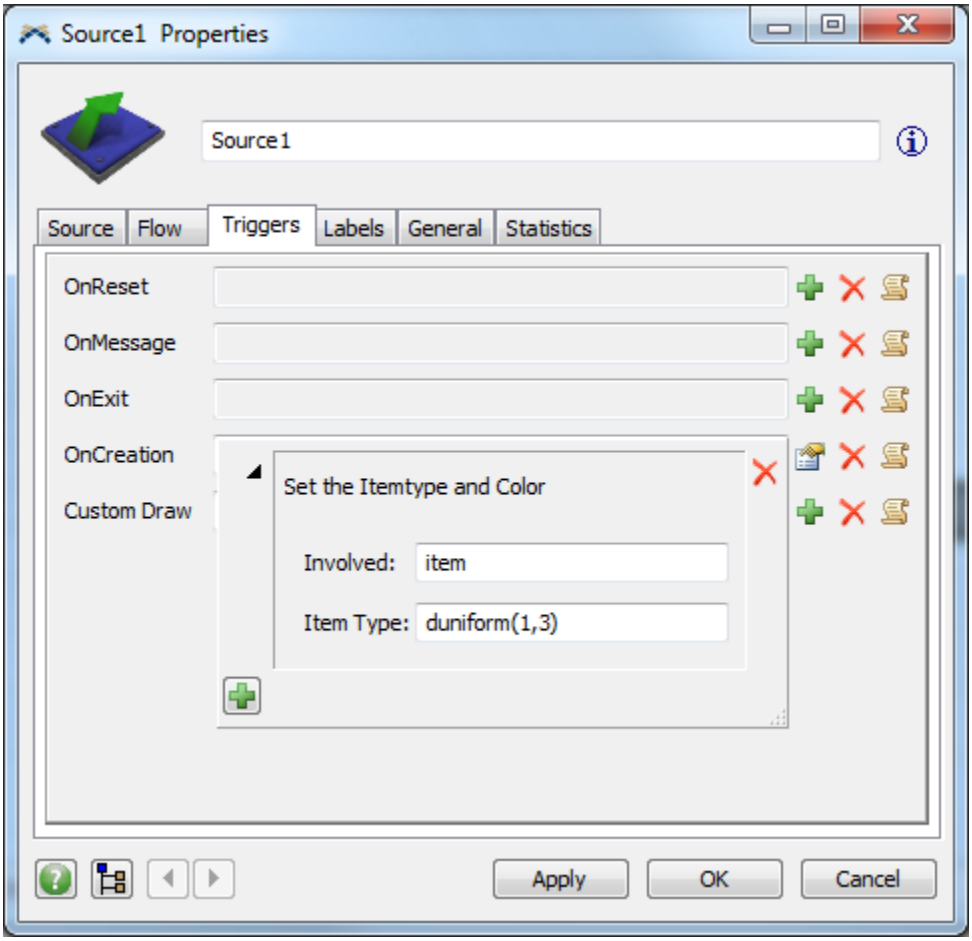
Kinematics can be used to do mutliple simultaneous movements to a single object. Try adding more *addkinematics* commands to the **OnSetupFinish** trigger.

Part 2: Update Kinematics Dynamically

In this part of the tutorial, we will have the processor spin at different speeds according to the item type.

Step 8: Create Multiple Itemtypes

- Open the properties window of the *Source*.
- Go to the **Triggers** tab.
- Click on the add button the next to the **OnCreation** trigger.
- Select the *Set Itemtype and Color* pick option.
- Leave the values as default.

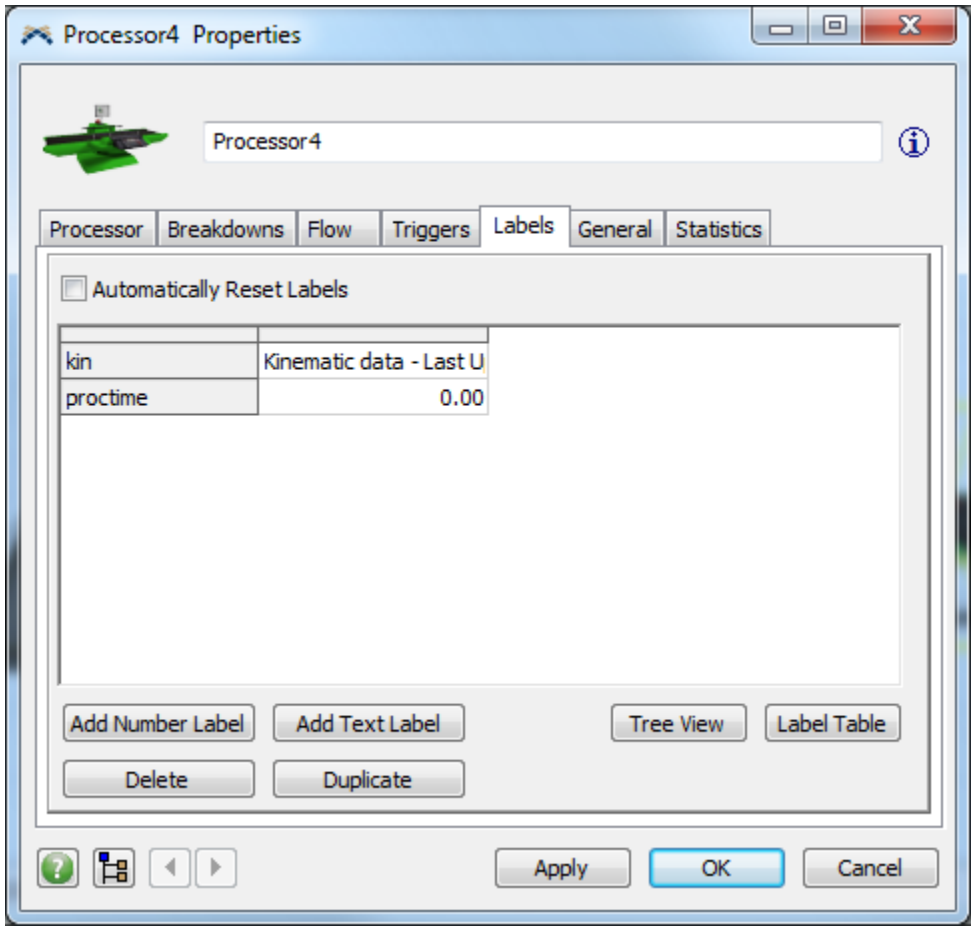


- Click **OK** to apply and close the properties window.

Step 9: Process Each Itemtype Differently

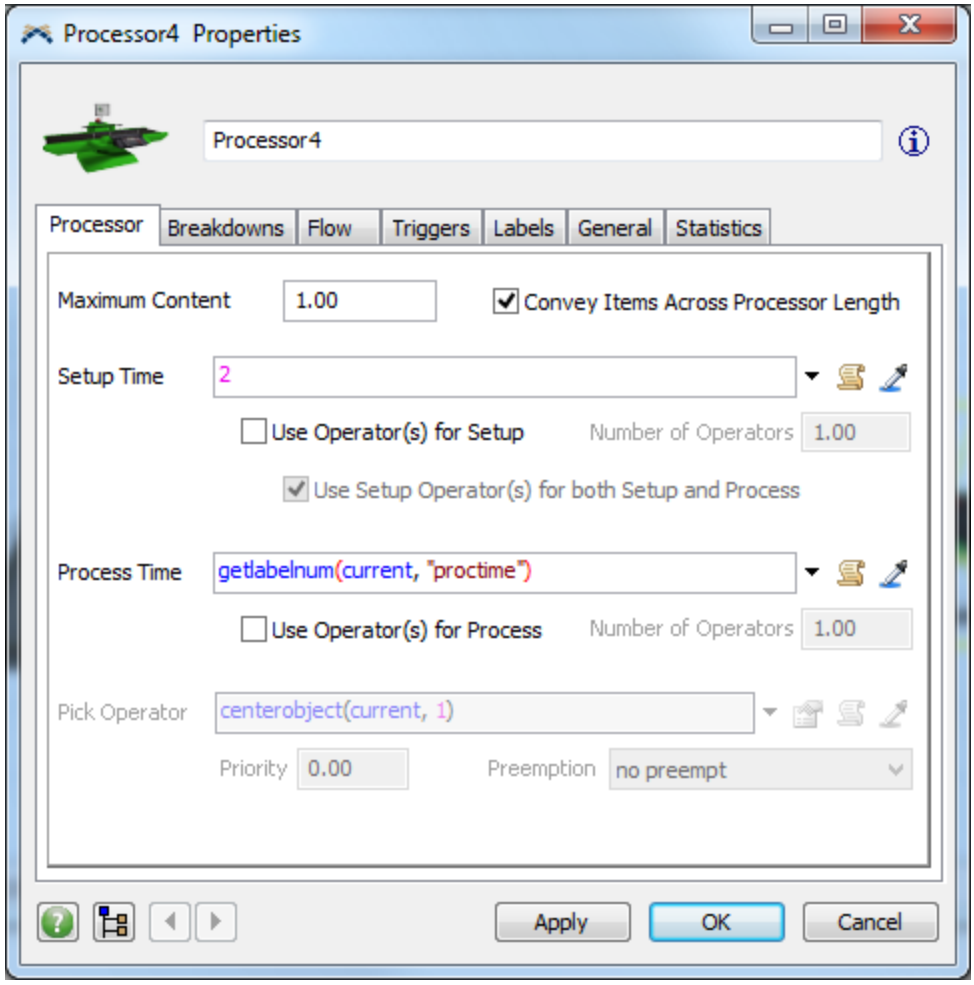
Each item type will be processed at a different speed and therefore require a different process time. Since this will change dynamically as the model runs, we will make a number label to keep track of the information.

- Open the properties window of the *Processor*.
- Go to the **Labels** tab.
- Click **Add Number Label** and rename the label *"proctime"*.



- Go to the **Processor** tab.
- Enter the following command into the **Process Time** field:

```
getlabelnum(current, "proctime")
```



Step 10: Create Dynamic Kinematics

We need to customize the kinematics so that it changes for each item type.

- Go to the **Triggers** tab.
- Click the code edit button for the **OnSetupFinish** trigger.
- Enter the following code:

```
initkinematics(label(current, "kin"), 13, 1, 0, 0, 0, 0, 0, 0);
int z;
int speed;
int type = getitemtype(item);
switch (type)
{
     case 1:
     z = 1080;
     speed = 360;
     break;

     case 2:
     z = 3240;
     speed = 360;
     break;
```

```
    case 3:
    z = 3240;
    speed = 180;
    break;
}
addkinematic(label(current, "kin"), 0, 0, z, speed, 90, 180, 0, 0, time(), 2);
```

```
/**Custom Code*/
treenode  item = parnode(1);
treenode  current = ownerobject(c);

initkinematics(label(current, "kin"), 13, 1, 0, 0, 0, 0, 0, 0);

int z;
int speed;

int type = getitemtype(item);

switch (type)
{
    case 1:
    z = 1080;
    speed = 360;
    break;

    case 2:
    z = 3240;
    speed = 360;
    break;

    case 3:
    z = 3240;
    speed = 180;
    break;
}

addkinematic(label(current, "kin"), 0, 0, z, speed, 90, 180, 0, 0, time(), 2);
```

The *initkinematics* command should be the same as in Part 1. The *addkinematics* command is placed within an if statement that checks the item type for each item as the setup finishes. Try playing with the z-rotation value as well as the maxspeed, acceleration, and deceleration values of the processor.

Step 11: Update the Process Time

We will now get the kinematic information from our object and use it to change the processing time of the processor.

- While still in the custom code for the **OnSetupFinish** trigger, add the following code:

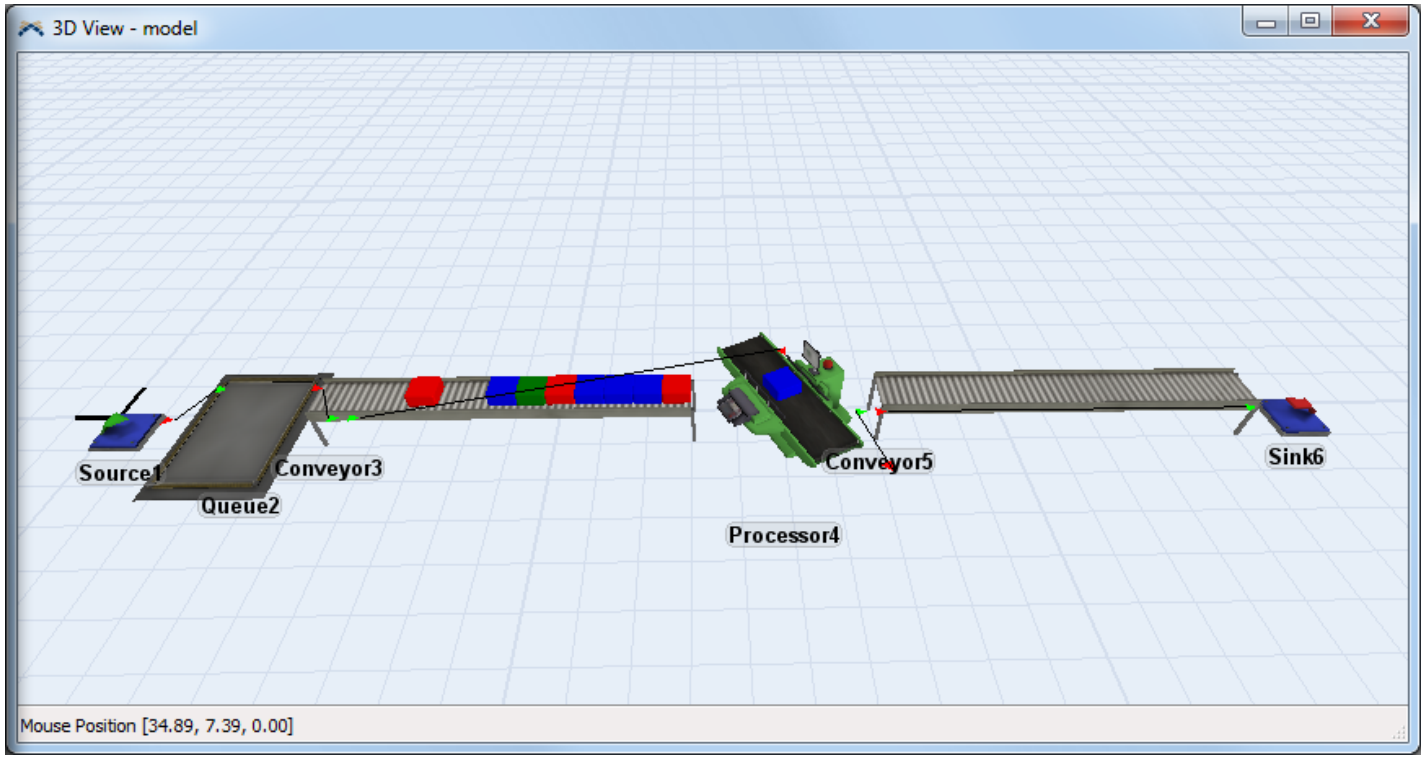
```
double endtime = getkinematics(label(current, "kin"), KINEMATIC_ENDTIME);
double starttime = getkinematics(label(current, "kin"), KINEMATIC_STARTTIME);
double proctime = endtime - starttime;
setlabelnum(current, "proctime", proctime);
```

The *getkinematics* command gets information from the last update made to the kinematic. The available information that can be grabbed is described in the Kinematics section of the help manual.

If you have more than one kinematic at a time, you can set which kinematic it pulls information from, and even get information at specific distances or travel times in the middle of the kinematic action. See the manual for details.

- Click **OK** to apply and close the code edit window.
- Click **OK** to apply and close the properties window.

Reset and run your model. Your model should look similar to this:



This completes the Kinematics tutorial. Congratulations!

Task Sequence Tutorial 1

1. Introduction
2. Step-By-Step Model Construction

Introduction

In this tutorial, you will learn how to build a basic task sequence from scratch. The operator will pick up the flowitem from a queue, take it to a table to inspect the item, then take the item to a processor. Writing your own task sequence will allow you to allocate and use one Operator for the entire task. This tutorial assumes a solid knowledge of basic interaction with the software, and will ask you to write several lines of code.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

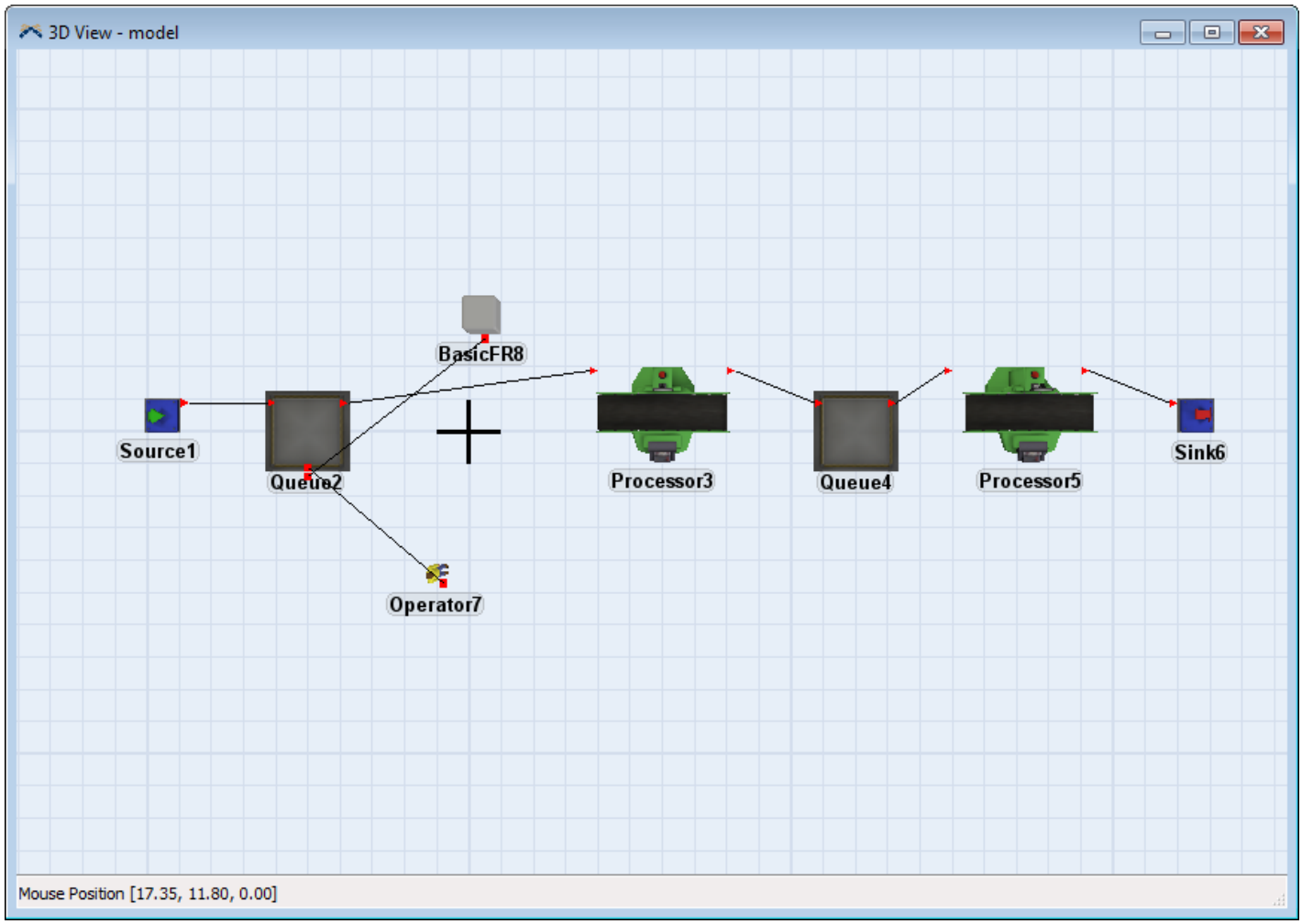
For more on Tasktypes and their parameters, see the Task Types page.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

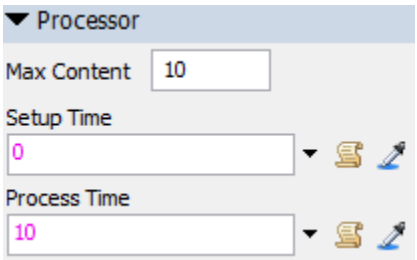
Step 1: Setup the model

- Create a **Source**, a **Queue**, a **BasicFR**, a **Processor**, a **Queue**, a **Processor**, an **Operator** and a **Sink**, and lay them out as shown in the picture below. The BasicFR will act as our table, but will not have any logic applied to it, and will not function in any way. It is just there to give the Operator somewhere to travel to. A Visual Tool could have also been used, or any of the Fixed Resources.
- Connect the objects as shown in the picture below, making sure to connect the Operator and the BasicFR to the center port of the first Queue in that order.

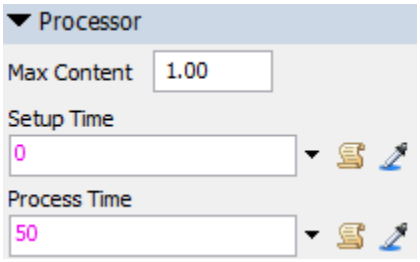


Step 2: Edit the Objects

- Click on the first Processor to open its properties in the Quick Properties.
- Under the the **Processor** section, change **Max Content** to **10**.



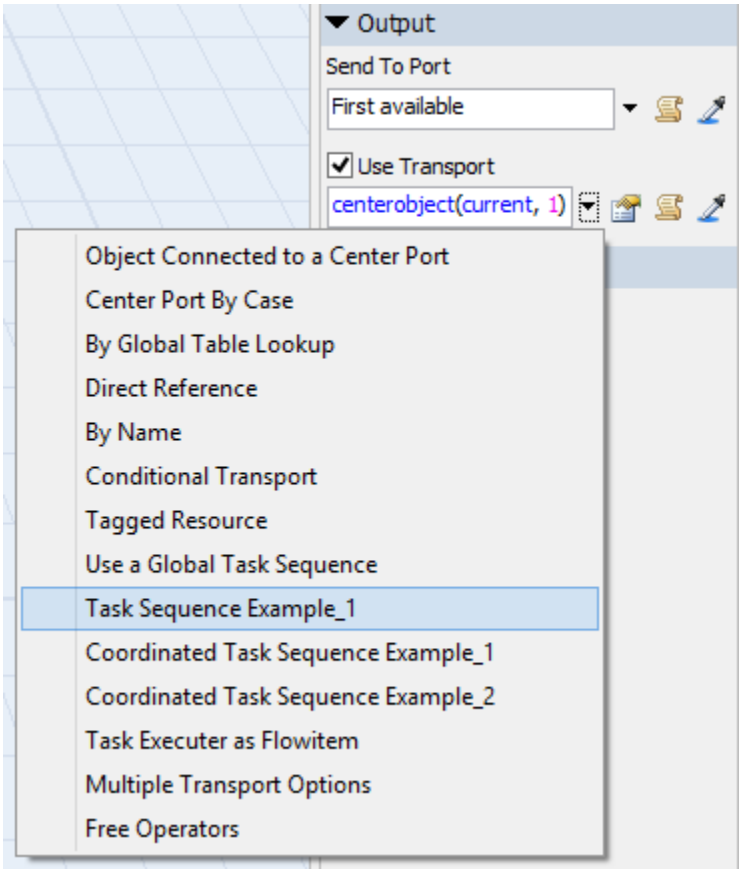
- Click on the first Processor to open its properties in the Quick Properties.
- Under the the **Processor** section, change **Process Time** to **50**.



Step 3: Write the Task Sequence

To make things easy, you will use the Basic Task Sequence Example to start the task sequence. From there, you will alter and add to it to fit your needs.

- Click the first Queue to open its properties in the Quick Properties.
- Under the the **Flow** section, check the **Use Transport** box, and select **Task Sequence Example_1** from the list. This task sequence example by default provides the same functionality as referencing an operator. The Operator travels to the current object, loads the item, travels to the downstream object, and unloads the item. You will alter this just a little bit.



- Click the **Code Edit** button to the right of the drop-down list to open the code editor. In this model, the Operator should do this task, and nothing else. So, you will remove the Break task by deleting all of the code on line 21. See the image below.

```
1 treenode item = parnode(1);
2 treenode current = ownerobject(c);
3 int port = parval(2);
4 /**Task Sequence Example 1*/
5 /**Creates a standard task sequence manually.*/
6 /**If this "Request Transport From" field returns a 0 rather than a valid pointer
7 to either a dispatcher or taskexecuter, then no call is made, and it is assumed
8 that the user will dispatch their own tasksequence.
9
10 This example shows the code that is required to create the exact same tasksequence
11 that is normally created automatically and dispatched to the object referenced by this fiel
12
13 treenode dispatcher = centerobject(current,1); // the dispatcher or task executer
14 double priority = getvarnum(current,"transportpriority"); // read the Priority value on the
15 int preempting = getvarnum(current,"preempttransport"); // read the Preemption mode on the
16
17 treenode ts = createemptytasksequence(dispatcher,priority,preempting);
18
19 inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
20 inserttask(ts,TASKTYPE_FRLOAD,item,current,port);
21
22 inserttask(ts,TASKTYPE_TRAVEL,outobject(current,port),NULL);
23 inserttask(ts,TASKTYPE_FRUNLOAD,item,outobject(current,port),opipno(current,port));
24
25 dispatchtasksequence(ts);
26 // return a 0 so this object will know that you made your own tasksequence and it doesn't n
27 //to make the standard tasksequence automatically
28 return 0;
```

After the Operator loads the item, we want him to Travel to the BasicFR, and Delay for 10 seconds before traveling to the downstream Processor.

- On the now empty line 21, type the following:
  `inserttask(ts, TASKTYPE_TRAVEL, centerobject(current,2), NULL);`
  Press the **Enter** key to go to the next line.

```
1 treenode item = parnode(1);
2 treenode current = ownerobject(c);
3 int port = parval(2);
4 /**Task Sequence Example 1*/
5 /**Creates a standard task sequence manually.*/
6 /**If this "Request Transport From" field returns a 0 rather than a valid pointer
7 to either a dispatcher or taskexecuter, then no call is made, and it is assumed
8 that the user will dispatch their own tasksequence.
9
10 This example shows the code that is required to create the exact same tasksequence
11 that is normally created automatically and dispatched to the object referenced by this fiel
12
13 treenode dispatcher = centerobject(current,1); // the dispatcher or task executer
14 double priority = getvarnum(current,"transportpriority"); // read the Priority value on the
15 int preempting = getvarnum(current,"preempttransport"); // read the Preemption mode on the
16
17 treenode ts = createemptytasksequence(dispatcher,priority,preempting);
18
19 inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
20 inserttask(ts,TASKTYPE_FRLOAD,item,current,port);
21 inserttask(ts,TASKTYPE_TRAVEL,centerobject(current,2),NULL);
22
23 inserttask(ts,TASKTYPE_TRAVEL,outobject(current,port),NULL);
24 inserttask(ts,TASKTYPE_FRUNLOAD,item,outobject(current,port),opipno(current,port));
25
26 dispatchtasksequence(ts);
27 // return a 0 so this object will know that you made your own tasksequence and it doesn't n
28 //to make the standard tasksequence automatically
```

- On line 22, type the following:
  `inserttask(ts, TASKTYPE_DELAY, NULL, NULL, 10, STATE_BUSY);`
  Click the **OK** button to close the code window.

```
treenode item = parnode(1);
treenode current = ownerobject(c);
int port = parval(2);
/**Task Sequence Example 1*/
/**Creates a standard task sequence manually.*/
/**If this "Request Transport From" field returns a 0 rather than a valid pointer
to either a dispatcher or taskexecuter, then no call is made, and it is assumed
that the user will dispatch their own tasksequence.

This example shows the code that is required to create the exact same tasksequence
that is normally created automatically and dispatched to the object referenced by this fiel

treenode dispatcher = centerobject(current,1); // the dispatcher or task executer
double priority = getvarnum(current,"transportpriority"); // read the Priority value on the
int preempting = getvarnum(current,"preempttransport"); // read the Preemption mode on the

treenode ts = createemptytasksequence(dispatcher,priority,preempting);

inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
inserttask(ts,TASKTYPE_FRLOAD,item,current,port);
inserttask(ts,TASKTYPE_TRAVEL,centerobject(current,2),NULL);
inserttask(ts,TASKTYPE_DELAY,NULL,NULL,10,STATE_BUSY);
inserttask(ts,TASKTYPE_TRAVEL,outobject(current,port),NULL);
inserttask(ts,TASKTYPE_FRUNLOAD,item,outobject(current,port),opipno(current,port));

dispatchtasksequence(ts);
// return a 0 so this object will know that you made your own tasksequence and it doesn't n
//to make the standard tasksequence automatically
```

- Click the **OK** button on the Properties window to close it.

Step 4: Reset and Run the Model

- **Reset and Run the model**. The Operator should Travel to the Queue, Load the item, Travel to the BasicFR, Delay for 10 seconds, Travel to the Processor, and Unload the item.
- **Save the model**. The next tutorial will build off of what you have done here.

Task Sequence Tutorial 2

1. Introduction
2. Step-By-Step Model Construction

Introduction

In this tutorial, you will build off the model you completed in Task Sequence Tutorial 1. The Operator will now stay at the Processor to process the item before he starts the next task sequence. This tutorial assumes a solid knowledge of basic interaction with the software, and will ask you to write several lines of code.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

For more on Tasktypes and their parameters, see the Task Types page.

Click here for the Step-By-Step Tutorial.

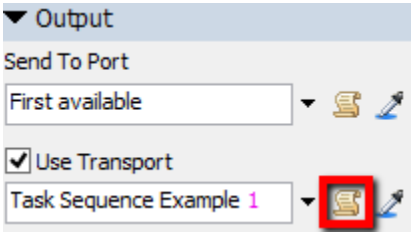
Step-By-Step Model Construction

Step 1: Load the Model

If you have not already done so, load the model from Task Sequence Tutorial 1.

Step 2: Add the Utilize Task

- Click on the first Queue to open its properties in the Quick Properties.
- Under the the **Flow** section, click the **Code Edit** button to the right of the **Use Transport** picklist to open the code editor.



- Add a new line after Line 24. On line 25, type the following:
  `inserttask(ts, TASKTYPE_UTILIZE, item, outobject(current, 1), STATE_UTILIZE);`

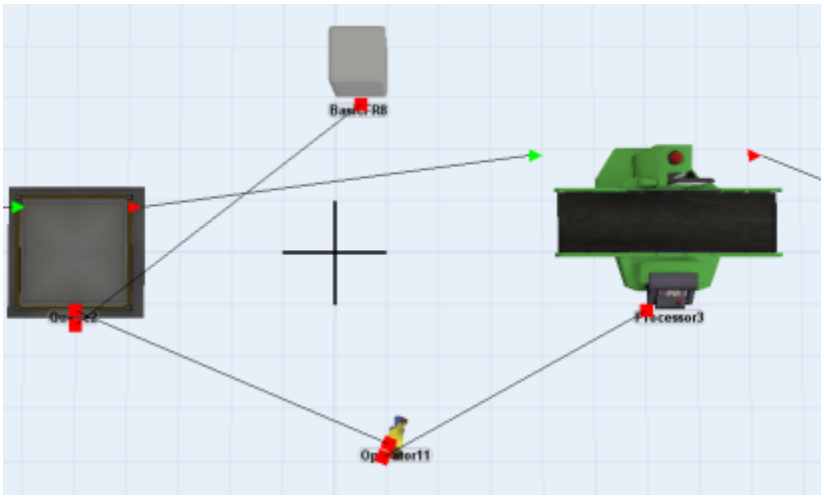
```
1  treenode item = parnode(1);
2  treenode current = ownerobject(c);
3  int port = parval(2);
4  /**Task Sequence Example 1*/
5  /**Creates a standard task sequence manually.*/
6  /**If this "Request Transport From" field returns a 0 rather than a valid pointer
7  to either a dispatcher or taskexecuter, then no call is made, and it is assumed
8  that the user will dispatch their own tasksequence.
9
10 This example shows the code that is required to create the exact same tasksequence
11 that is normally created automatically and dispatched to the object referenced by this fiel
12
13 treenode dispatcher = centerobject(current,1); // the dispatcher or task executer
14 double priority = getvarnum(current,"transportpriority"); // read the Priority value on the
15 int preempting = getvarnum(current,"preempttransport"); // read the Preemption mode on the
16
17 treenode ts = createemptytasksequence(dispatcher,priority,preempting);
18
19 inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
20 inserttask(ts,TASKTYPE_FRLOAD,item,current,port);
21 inserttask(ts,TASKTYPE_TRAVEL,centerobject(current,2),NULL);
22 inserttask(ts,TASKTYPE_DELAY,NULL,NULL,10,STATE_BUSY);
23 inserttask(ts,TASKTYPE_TRAVEL,outobject(current,port),NULL);
24 inserttask(ts,TASKTYPE_FRUNLOAD,item,outobject(current,port),opipno(current,port));
25 inserttask(ts,TASKTYPE_UTILIZE,item,outobject(current,1),STATE_UTILIZE);
26
27 dispatchtasksequence(ts);
28 // return a 0 so this object will know that you made your own tasksequence and it doesn't n
```

- Click **OK** to close the Code Edit window.

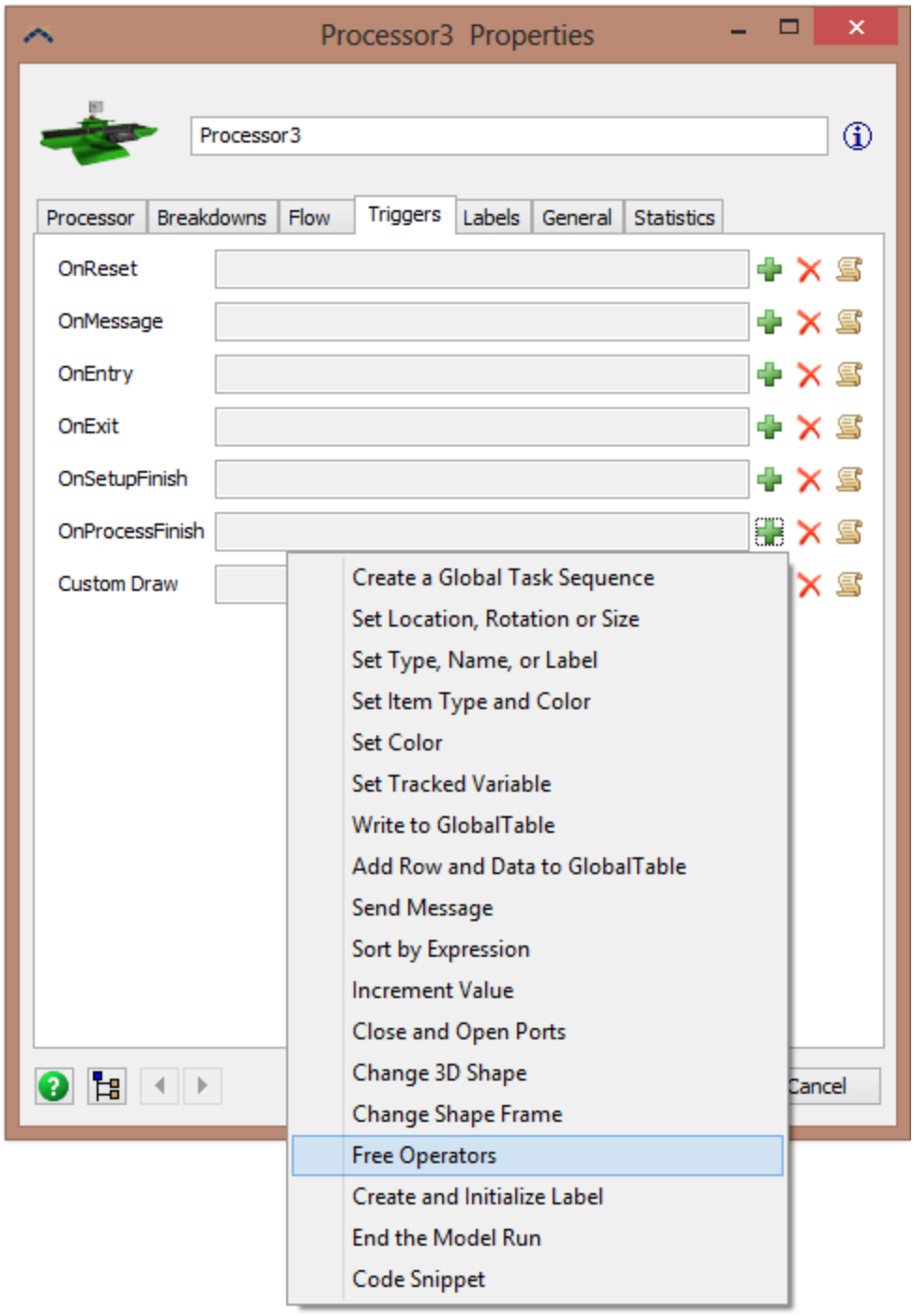
If you Run the model now, you will notice that the operator stays at the Processor forever after the item has been unloaded and processed. This is because nothing is freeing the Operator, and the Operator will stay utilized until it is freed by an object. The best place to do this is in the OnProcessFinish of the Processor that is utilizing the Operator.

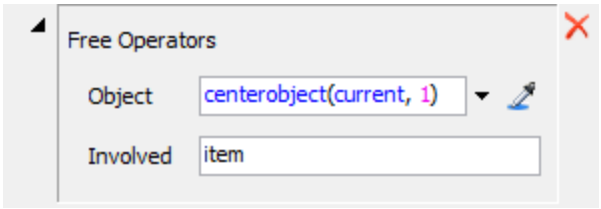
Step 3: Edit the Processor to Free the Operator

- Connect the Operator to the first Processor with a center port connection.



- Double-click the first Processor to open its Properties window, then click the **Triggers** tab. In the **OnProcessFinish** trigger list, select **Free Operators**. The default trigger parameters will work for this model. The Involved is the first parameter in the TASKTYPE_UTILIZE command we used in the previous step. In order for the Operator to be freed, the involved object must match, in this case, item.





- Click the **OK** button on the Properties window to close it.

Step 4: Reset and Run the Model

- **Reset and Run the model**. The Operator should Travel to the Queue, Load the item, Travel to the BasicFR, Delay for 10 seconds, Travel to the Processor, Unload the item, and stay at the Processor for the Process Time.

- **Save the model**. The next tutorial will build off of what you have done here.

Task Sequence Tutorial 3

1. Introduction
2. Step-By-Step Model Construction

Introduction

In this tutorial, you will build off the model you completed in Task Sequence Tutorial 2. The Operator will now pick the item up from the Processor, and carry it to the second Queue. This tutorial assumes a solid knowledge of basic interaction with the software, and will ask you to write several lines of code.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

For more on Tasktypes and their parameters, see the Task Types page.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

Step 1: Load the Model

If you have not already done so, load the model from Task Sequence Tutorial 2.

Step 2: Delete the OnProcessFinish Trigger

Since you will be adding to the task sequence, a change needs to be made in how the Operator is released from the Utilize task. The freeoperator() command needs to be moved to the Request Transport From field in order for the model to work correctly. If you need more information, an in-depth explanation will be provided at the end of the tutorial.

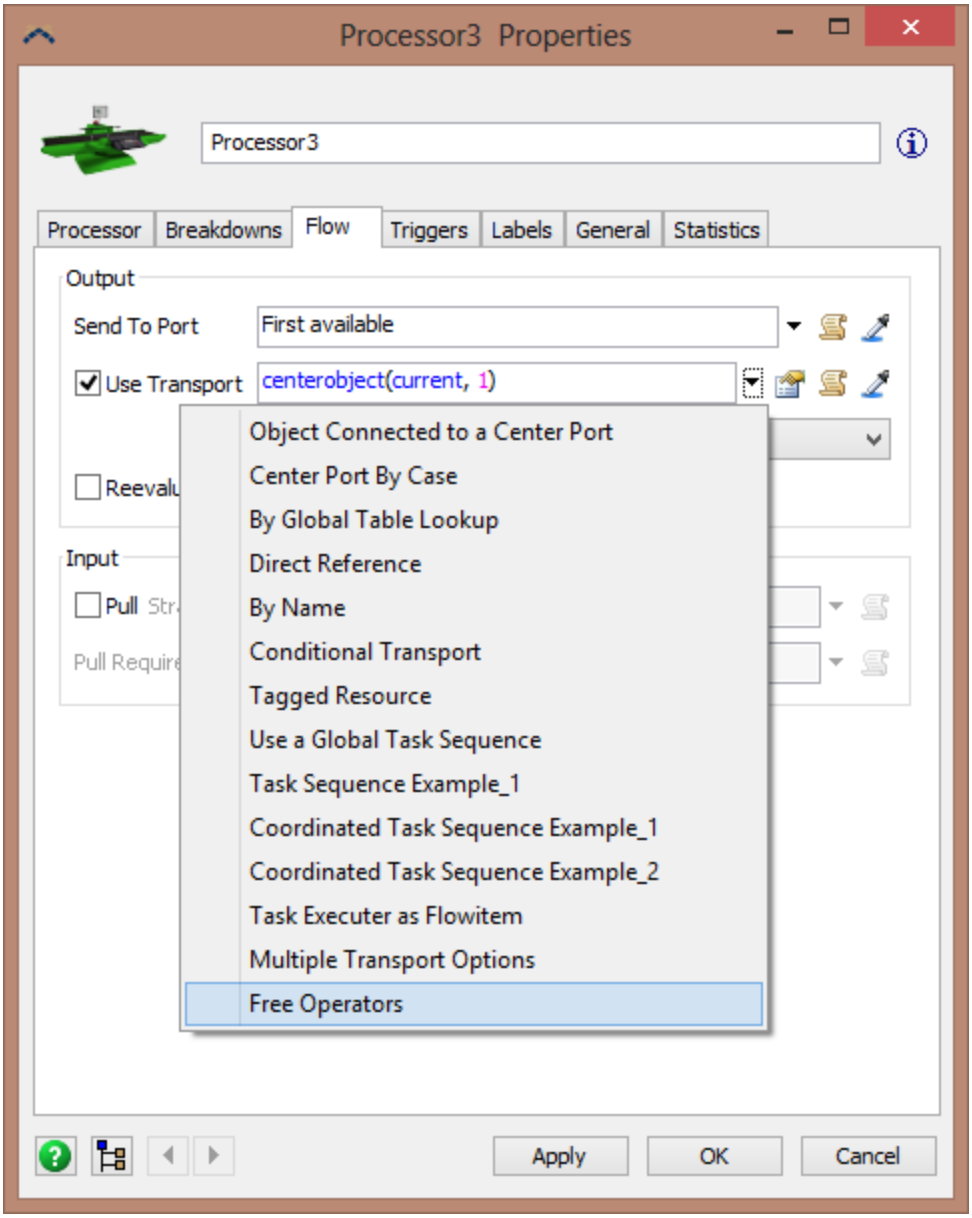
- Double-click on the first Processor to open its Properties window, and click the **Triggers** tab.
- In the **OnProcessFinish** trigger, click the button, and then click the button to remove the function on this trigger.
- Don't close the Properties window yet.

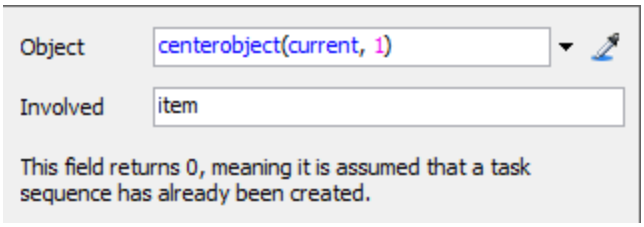
Step 3: Write the Flow Logic

Now you will need to free the operator in the Request Transport From ( ) field. Also, since the next set of tasks you will add will override the Processor's flow logic, specifically the Request Transport From logic, you need to tell the processor that it doesn't need to create a task sequence.

Note: Whenever you write a task sequence that controls the output of an object somewhere other than the **Request Transport From** on the object the task sequence is affecting, you **MUST** return 0 in that object's Request Transport From logic, otherwise there will be serious problems. In this example, you are writing the task sequence on the Queue, but the next tasks you will add affect the Processors natural transport logic, so even though the task sequence is on the Queue, you will need to return 0 on the Processor's Request Transport From logic. You would also need to do this if you were writing a task sequence in a trigger field instead of in the Request Transport From.

- Double-click on the first Processor to open its Properties window, and click the **Flow tab**.
- Check the **Use Transport** box, then choose **Free Operators** from the drop down list. The default parameters should work fine for this model. Notice that the description points out that this option returns 0 for you, so you don't have to worry about writing this anywhere.

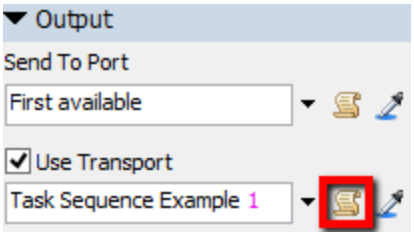



Object
centerobject(current, 1)
Involved
item
This field returns 0, meaning it is assumed that a task sequence has already been created.

- Click the **OK** button to close the Properties window.

Step 3: Write the rest of the Task Sequence

- Click on the first Queue to open its properties in the Quick Properties.
- Under the the **Flow** section, click the **Code Edit** button to the right of the **Use Transport** picklist to open the code editor.



- You will need to create a local variable so that you can more easily reference the second Queue in you task sequence. On line 17, type the following:
  `treenode downQueue = outobject(outobject(current, 1), 1);`

```
 1 treenode item = parnode(1);
 2 treenode current = ownerobject(c);
 3 int port = parval(2);
 4 /**Task Sequence Example 1*/
 5 /**Creates a standard task sequence manually.*/
 6 /**If this "Request Transport From" field returns a 0 rather than a valid pointer
 7 to either a dispatcher or taskexecuter, then no call is made, and it is assumed
 8 that the user will dispatch their own tasksequence.
 9
10 This example shows the code that is required to create the exact same tasksequence
11 that is normally created automatically and dispatched to the object referenced by this fiel
12
13 treenode dispatcher = centerobject(current,1); // the dispatcher or task executer
14 double priority = getvarnum(current,"transportpriority"); // read the Priority value on the
15 int preempting = getvarnum(current,"preempttransport"); // read the Preemption mode on the
16
17 treenode downQueue = outobject(outobject(current,1),1);
18 treenode ts = createemptytasksequence(dispatcher,priority,preempting);
19
20 inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
21 inserttask(ts,TASKTYPE_FRLOAD,item,current,port);
22 inserttask(ts,TASKTYPE_TRAVEL,centerobject(current,2),NULL);
23 inserttask(ts,TASKTYPE_DELAY,NULL,NULL,10,STATE_BUSY);
24 inserttask(ts,TASKTYPE_TRAVEL,outobject(current,port),NULL);
25 inserttask(ts,TASKTYPE_FRUNLOAD,item,outobject(current,port),opipno(current,port));
26 inserttask(ts,TASKTYPE_UTILIZE,item,outobject(current,1),STATE_UTILIZE);
27
28 dispatchtasksequence(ts);
```

- Starting on line 27, type the following:
  `inserttask(ts, TASKTYPE_FRLOAD, item, outobject(current, 1));`
  `inserttask(ts, TASKTYPE_TRAVEL, downQueue, NULL);`
  `inserttask(ts, TASKTYPE_FRUNLOAD, item, downQueue, 1);`

```
/**Creates a standard task sequence manually.*/
/**If this "Request Transport From" field returns a 0 rather than a valid pointer
to either a dispatcher or taskexecuter, then no call is made, and it is assumed
that the user will dispatch their own tasksequence.

This example shows the code that is required to create the exact same tasksequence
that is normally created automatically and dispatched to the object referenced by this fiel

treenode dispatcher = centerobject(current,1); // the dispatcher or task executer
double priority = getvarnum(current,"transportpriority"); // read the Priority value on the
int preempting = getvarnum(current,"preempttransport"); // read the Preemption mode on the

treenode downQueue = outobject(outobject(current,1),1);
treenode ts = createemptytasksequence(dispatcher,priority,preempting);

inserttask(ts,TASKTYPE_TRAVEL,current,NULL);
inserttask(ts,TASKTYPE_FRLOAD,item,current,port);
inserttask(ts,TASKTYPE_TRAVEL,centerobject(current,2),NULL);
inserttask(ts,TASKTYPE_DELAY,NULL,NULL,10,STATE_BUSY);
inserttask(ts,TASKTYPE_TRAVEL,outobject(current,port),NULL);
inserttask(ts,TASKTYPE_FRUNLOAD,item,outobject(current,port),opipno(current,port));
inserttask(ts,TASKTYPE_UTILIZE,item,outobject(current,1),STATE_UTILIZE);
inserttask(ts,TASKTYPE_FRLOAD,item,outobject(current,1));
inserttask(ts,TASKTYPE_TRAVEL,downQueue,NULL);
inserttask(ts,TASKTYPE_FRUNLOAD,item,downQueue,1);

dispatchtasksequence(ts);
// return a 0 so this object will know that you made your own tasksequence and it doesn't n
```

- Click the **OK** button on the Code Window and theProperties window to close them.

Note: The reason that the freeoperators() command needed to be changed from the OnProcessFinish is due to the fact that a written task sequence has the ability to override the internal logic of objects. If the downstream queue isn't available when the Processor finishes its process time, then the Processor will release the Operator to load/unload the part into the Queue before it's actually available, which can cause the Queue to be over-filled, and can cause some other problems. So by moving the freeoperators() to the Request Transport From field, the operator will only be freed to continue with the load/unload only when the downstream queue is ready to receive that part. The perfect solution to this problem would be to have two utilize tasks, freeing the Operator on both the OnProcessFinish, and the Request Transport From. This way, the operator would be free after the Processor finishes, in case you need him to break, but he won't move the item until the downstream object is available. See the Fixed Resource page for more information.

Step 4: Reset and Run the Model

- **Reset and Run the model**. The Operator should Travel to the Queue, Load the item, Travel to the BasicFR, Delay for 10 seconds, Travel to the Processor, Unload the item, stay at the Processor for the Process Time, Load the item, Travel to the next Queue, and Unload the item.
- **Save the model**.

SQL Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

This tutorial will help you understand how to connect FlexSim to SQL databases.

Note: This is not a beginner tutorial, it is assumed you know basic FlexSim and are familiar with SQL databases. I will be using phpMyAdmin for SQL during the tutorials.

What You Will Learn

- Send information to a SQL database from FlexSim
- Read data from a SQL database for dynamic use within FlexSim

Approximate Time to Complete this Lesson

This lesson should take about 15-20 minutes to complete.

Model Overview

In this model we will get data from a table in a SQL database to determine process times. We will also send information to another table in the same database about flowitem staytimes.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

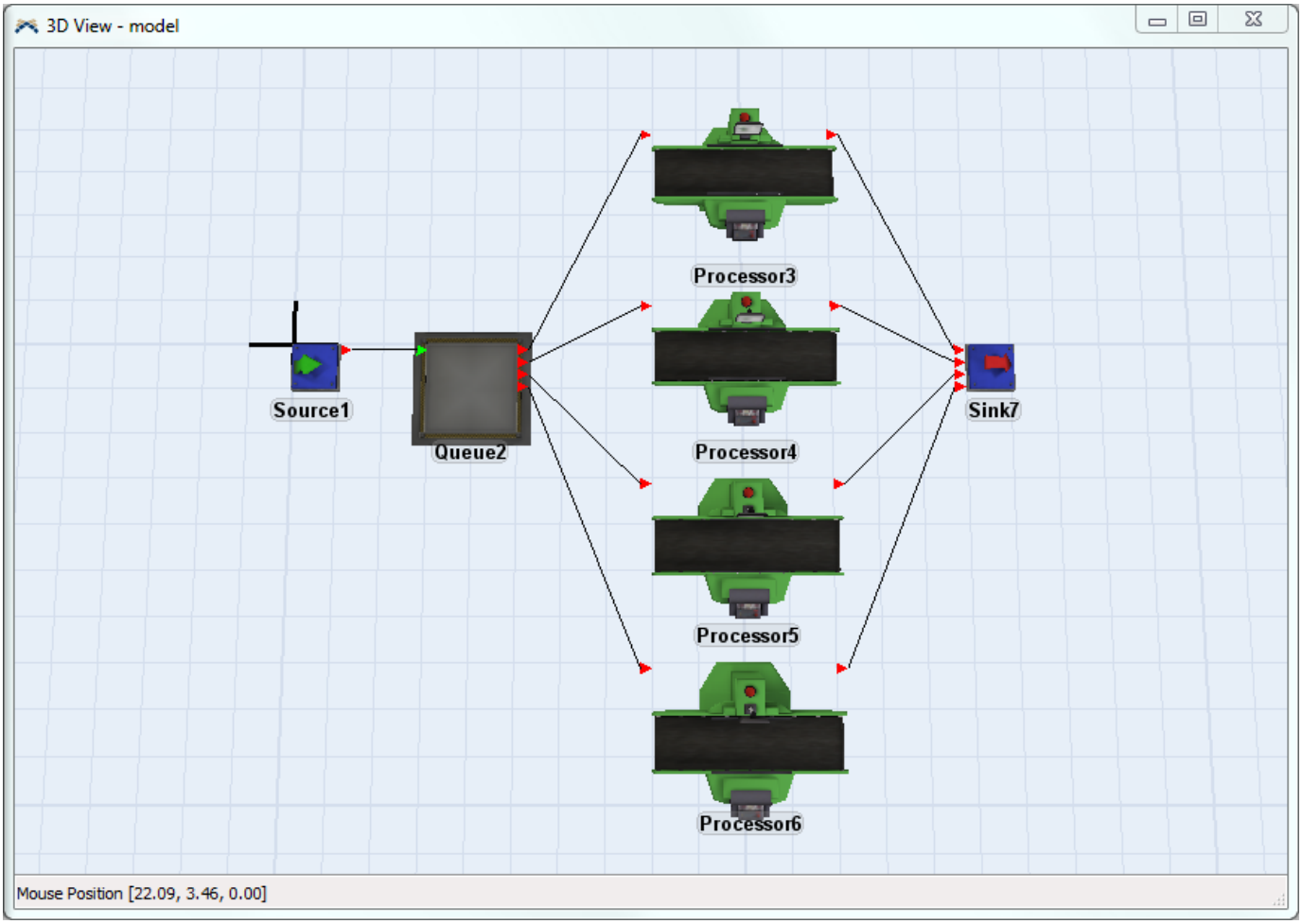
Building SQL Model

Begin a new model by clicking the button on the toolbar. Click OK on the Model Units window, we will use the default units for our model.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

Step 1: Create the Objects

- Drag objects from your Library Icon Grid onto the 3D View to create the model shown below.



Connect all of the objects as shown:

- Connect *Source1* to *Queue2*.
- Connect *Queue2* to *Processor3*, *Processor4*, *Processor5* and *Processor6*.
- Connect *Processor3*, *Processor4*, *Processor5* and *Processor6* to *Sink7*.

Step 2: Setup the SQL Database

We will create an SQL database that we can both read from and write to from our FlexSim model. First, we'll create a table that stores the process times for each processor based on itemtype.

- Create an SQL database called *"flexsimdata"*

- Create a table within *"flexsimdata"* called *"processtimes"* with 4 columns named *"Proc1"*, *"Proc2"*, *"Proc3"*, and *"Proc4"*. Each of these columns should contain **FLOAT** number data.
- Fill the columns as shown in the following figure:

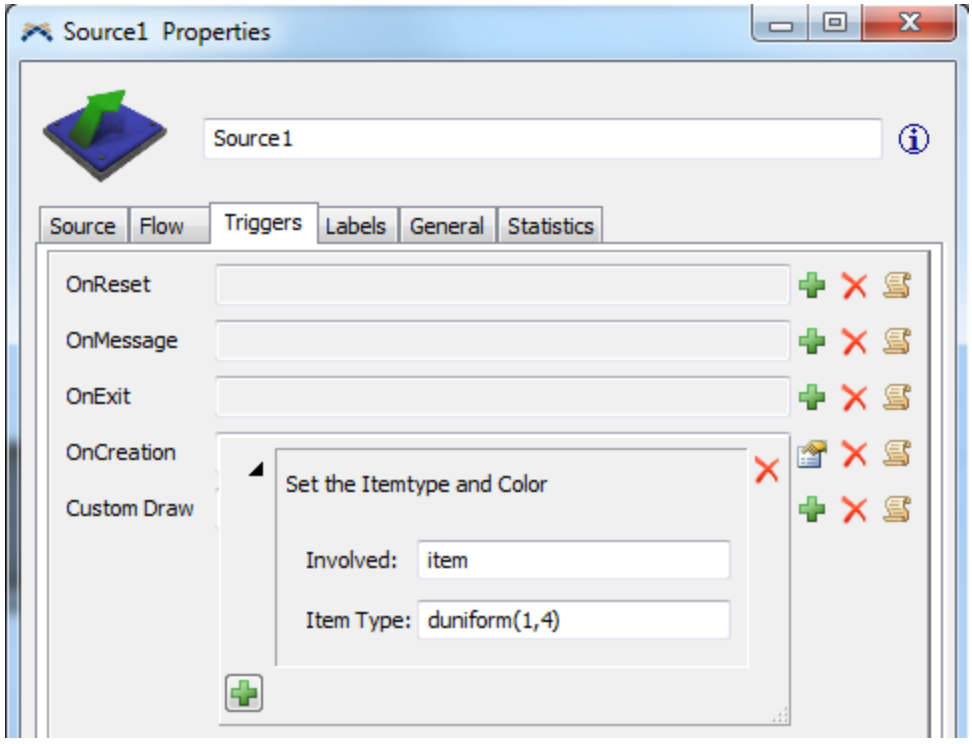
Proc1	Proc2	Proc3	Proc4
3.2	3.4	3.6	3.8
4.2	4.4	4.6	4.8
5.2	5.4	5.6	5.8
6.2	6.4	6.6	6.8

Step 3: Setup the Source

We will have the *Source* create 4 different itemtypes.

- Change the *Source's* **Inter-Arrivaltime** to *exponential(0, 3, 0)*

- Go to the **Triggers** tab.
- Click the add button for the **OnCreation** trigger.
- Select the **Set ItemType and Color** picklist option.
- Change **Item Type** to *duniform(1, 4)*



- Click **OK** to apply and close the Properties window.

Step 4: Setup the Processors

Each of the 4 processor's will access the SQL database we just created in order to set their processing time.

- Open one of the *Processor's* properties window.
- Click the code edit button next to *Process Time*.

- Enter the following code:

```
/**Custom Code*/
treenode current = ownerobject(c);
treenode item = parnode(1);

dbopen("flexsimdata", "processtimes", 1);

double proctime = dbgettablenum(getitemtype(item), 1);

dbclose();

return proctime;
```

- The **dbopen()** command will access your database and open it up for reading or writing. The first parameter in the name of the database as named in your Windows ODBC Data Source Administrator (not necessarily the name shown in phpMyAdmin). The second parameter in this case is the table you want to read from or write to. The third parameter toggles between table mode (1) and SQL mode (0). We use 1 in this case because we aren't using direct SQL queries, we are declaring a table to work with in the database.
- We use a special command called **dbgettablenum()** to get information from a table in a database. In our case, we want the row to match the item type, and we want each column to represent each processor. You will change the 1 to 2, 3, and 4 for the other processors.
- You need to use **dbclose()** to close the database so other databases can be accessed later on.

- Click **OK** to apply and close the Properties window.

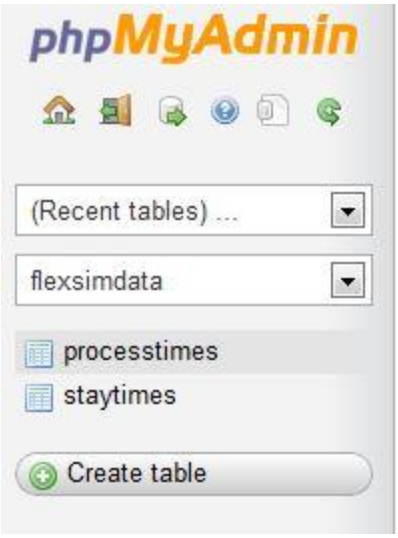
Write the same code on the other three processors, replacing each consecutive processor's second parameter in **dbgettablenum()** to 2, 3, and 4, respectively.

Reset and Run the model. If you look at the staytime statistic, you should notice that they correspond to the values in the *flexsimdata* database's *processtimes* table.

Step 5: Add Staytimes Table to Database

We will now add another table to our SQL database to store the stay times of all the items going through our model.

- Create a table within *"flexsimdata"* called *"staytimes"* with 2 columns named *"ID"* and *"Staytime"*. The first column should contain **integer** data and should autoincrement, and the second column should contain **FLOAT** number data.



- Create one initial row as shown in the following figure with *ID = 1* and *Staytime = 0*.

Step 6: Write to the SQL Database

Each time an item enters the Sink, we will record that item's time in the system (staytime) and write that value to the SQL database.

- Open the *Sink's* properties window.
- Go to the **Triggers** tab.
- Click the code edit button button for the **OnEntry** trigger.
- Enter the following code:

```
string staytime = numtostring(time() - get(stats_creationtime(item)),2, 3);
string query = concat("INSERT INTO  `flexsimdata`.`staytimes` (`Staytime`)VALUES ('",
staytime, "');");
string altquery = concat("UPDATE staytimes SET Staytime =", staytime, " WHERE ID =",
numtostring(getinput(current)));
dbopen("flexsimdata", "SELECT * FROM staytimes", 0);
int rows = dbgetnumrows();
if (getinput(current) < rows)
{
    dbsqlquery(altquery);
}
else
{
    dbsqlquery(query);
}
dbclose();
```

```
/**Custom Code*/
treenode item = parnode(1);
treenode current = ownerobject(c);
int port = parval(2);

string staytime = numtostring(time() - get(stats_creationtime(item)),2, 3);

string query = concat("INSERT INTO  `flexsimdata`.`staytimes` (`Staytime`)VALUES ('", staytime, "');");
string altquery = concat("UPDATE staytimes SET Staytime =", staytime, " WHERE ID =", numtostring(getinp

dbopen("flexsimdata", "SELECT * FROM staytimes", 0);
int rows = dbgetnumrows();
if (getinput(current) < rows)
{
    dbsqlquery(altquery);
}
else
{
    dbsqlquery(query);
}
dbclose();

```

Once we get the stay time into a variable called *staytime*, we make a query for adding data into the **Staytime** column of the *staytimes* table. The alternate query is made as well in case the table already contains data and needs to be over-written. Therefore, we have the if statement requiring us to use the alternate query for as long as there are previously existing rows.

- Click **OK** to apply and close the Properties window.

Reset and Run the model. You should notice the *staytimes* table of your database filling up with the information gathered at the *Sink*, similar to the following figure.

+ Options

	ID	Staytime
Edit Copy Delete	1	4.8
Edit Copy Delete	2	6.6
Edit Copy Delete	3	6.8
Edit Copy Delete	4	6.4
Edit Copy Delete	5	3.6
Edit Copy Delete	6	4.8
Edit Copy Delete	7	3.2
Edit Copy Delete	8	4.6
Edit Copy Delete	9	6.4
Edit Copy Delete	10	6.8

This completes the SQL tutorial. Congratulations!

Fluid Objects Tutorial

1. Introduction
2. Step-By-Step Model Construction

Introduction

This lesson introduces most of FlexSim's Fluid Objects. You will learn how they interact with each other and how to include them in a model with the Discrete Objects. Building a model with the Fluid Objects is more involved and requires more attention to detail than a model with the Discrete Objects. For that reason, it is recommended that you feel comfortable building models with the other objects before you begin to learn about the Fluid Objects.

What You Will Learn

- How to model fluid material with FlexSim
- How to convert flowitems into fluid material
- How to transfer and store fluid material
- How to use level marks on a tank to control material flow
- How to mix fluid materials together
- How to convert fluid material into flowitems

New Objects

In this lesson you will be introduced to the FluidTicker, ItemToFluid, FluidPipe, FluidTank, FluidMixer, FluidProcessor and FluidToItem objects.

Approximate Time to Complete this Lesson

This lesson should take about 45-60 minutes to complete.

Fluid Model Overview

In our fluid model we will have an operator carry boxes of two different types of material into the model. These boxes will be converted into two fluids which will be transported by Pipes to two Tanks. From the Tanks the material is sent to a single Mixer which will mix the two products into a new product. That product is sent through a FluidProcessor, and then converted into flowitems which are carried by a Conveyor to a Sink. The fluid in this model will be measured in gallons, and the time will be in seconds.

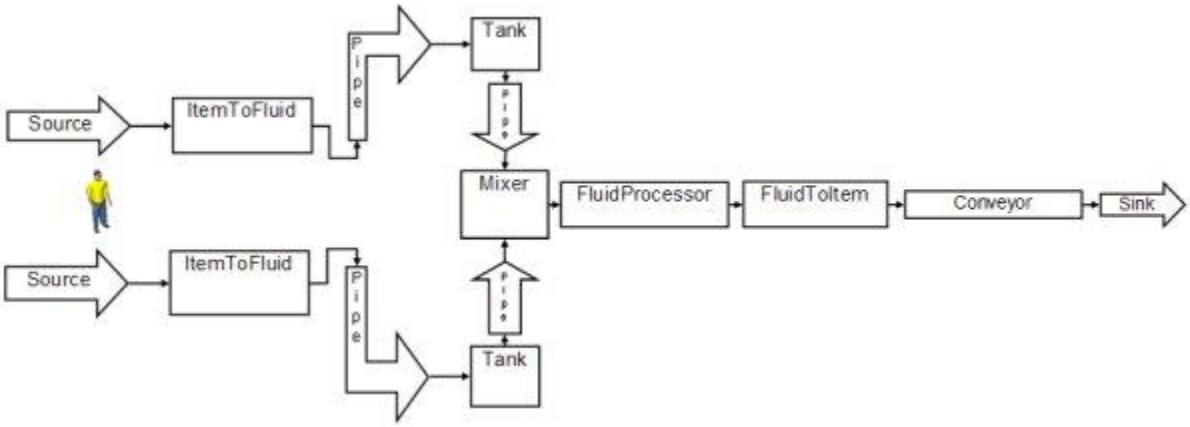


Figure 4-1 Fluid Model diagram

Fluid Model Data

Flowitem arrival rate: exponential(0,10) seconds
Maximum Content of ItemToFluid: 20 gallons
Fluid Units per Discrete Unit (ItemToFluid): 10 gallons per flowitem
Maximum Content of Pipe leading to Tank: 20 gallons
Transfer Rate (ItemToFluid to Tank): 2 gallons per second
Tank Low Mark: 1 gallon
Tank High Mark: 45 gallons
Maximum Content of Pipe leading to Mixer: 10 gallons

Transfer Rate (Tank to FluidToItem): 1 gallon per second

Mixer Steps:
Step 1: Material1, no delay time
Step 2: Material2, 10 second delay
Mixer Recipe:
Material1: 10 gallons, step 1
Material2: 20 gallons, step 2
Maximum Content of FluidToItem: 10 gallons
Fluid Units per Discrete Unit (FluidToItem): 10 gallons per flowitem

New Concepts

FlexSim Terminology

Before you start this model it will be helpful to understand some of the basic terminology of FlexSim's fluid system.

Fluid: Any material that is not easily or efficiently modeled with discrete flowitems. Typically, material that is measured by weight or volume is hard to model with flowitems. This is because frequently part of a unit (for example, half a gallon) can be moved by itself. There is no easy mechanism for moving half of a flowitem. Fluid material can also represent objects that are so numerous that flowitems are impractical. For example, thousands of bottles in a filling line will slow down a model that uses a flowitem for each bottle. Instead, the Fluid Objects can be used to model these bottles without the overhead that comes with the flowitems.

Fluid Objects: The eleven objects that are designed to handle fluid material. Nine of them cannot interact with FlexSim's Discrete objects, but two of them are designed to work as an interface between the Fluid Objects and the Discrete Objects. More information can be found here.

Tick: The Fluid Objects send and receive material at set intervals. These intervals are called "ticks". At the end of each tick, the Fluid Objects calculate how much material they sent and received during that time period.

Tick time: The length of each tick. The modeler can set this value to some value that is appropriate for their model. A shorter tick time may make the model more accurate, but it may also make it slower. A longer value will be a faster model, but the cost is a loss in accuracy. It is up to each modeler to decide the optimal trade-off of speed and accuracy for their model.

Rate: The maximum speed at which material enters or leaves an object. Generally, the Fluid objects have both an input rate and an output rate that are separate from each other. In a few objects, the rate at which material enters will affect the rate at which it leaves. For these objects, the modeler is not given the opportunity to edit the output rate. The actual rate at which material enters or leaves is based on several factors: the output rate of the upstream object, the input rate of the downstream object, the amount of material available to send and the amount of space available in the downstream object.

Object Rate: This is the maximum rate at which material can enter or leave an object through all input or output ports combined. The objects typically have a separate rate for the input ports and the output ports. If, at the end of any tick, the object calculates that the amount of material it has sent or received has reached the maximum object rate, no more material will be sent or received for that tick, even if there are ports that have not yet sent or received material.

Port Rate: This is the maximum rate at which material can enter or leave any single port on the object. The objects typically have different port rates for input and output ports. This single value applies to all of the input or output ports. It cannot be changed to affect individual ports.

Port Scale Factor: This is a number that is used to change the port rate for each individual port. There is one scale factor available for every input and output port. The value for each port is multiplied by the maximum port rate to find the actual maximum rate for that port.

Click here for the Step-By-Step Tutorial.

Step-By-Step Model Construction

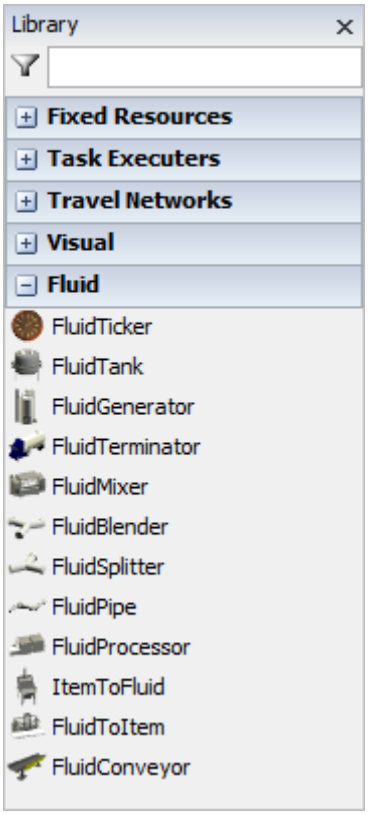
Building the Fluid Model

To start building the fluid model, you will need to start with a new model. Do not begin with the models you saved from the previous lessons. It is expected that you know how to create objects, connect their ports, and use their Properties GUIs.

If at any time you encounter difficulties while building this model, a fully functional tutorial model can be found at http://www.flexsim.com/tutorials

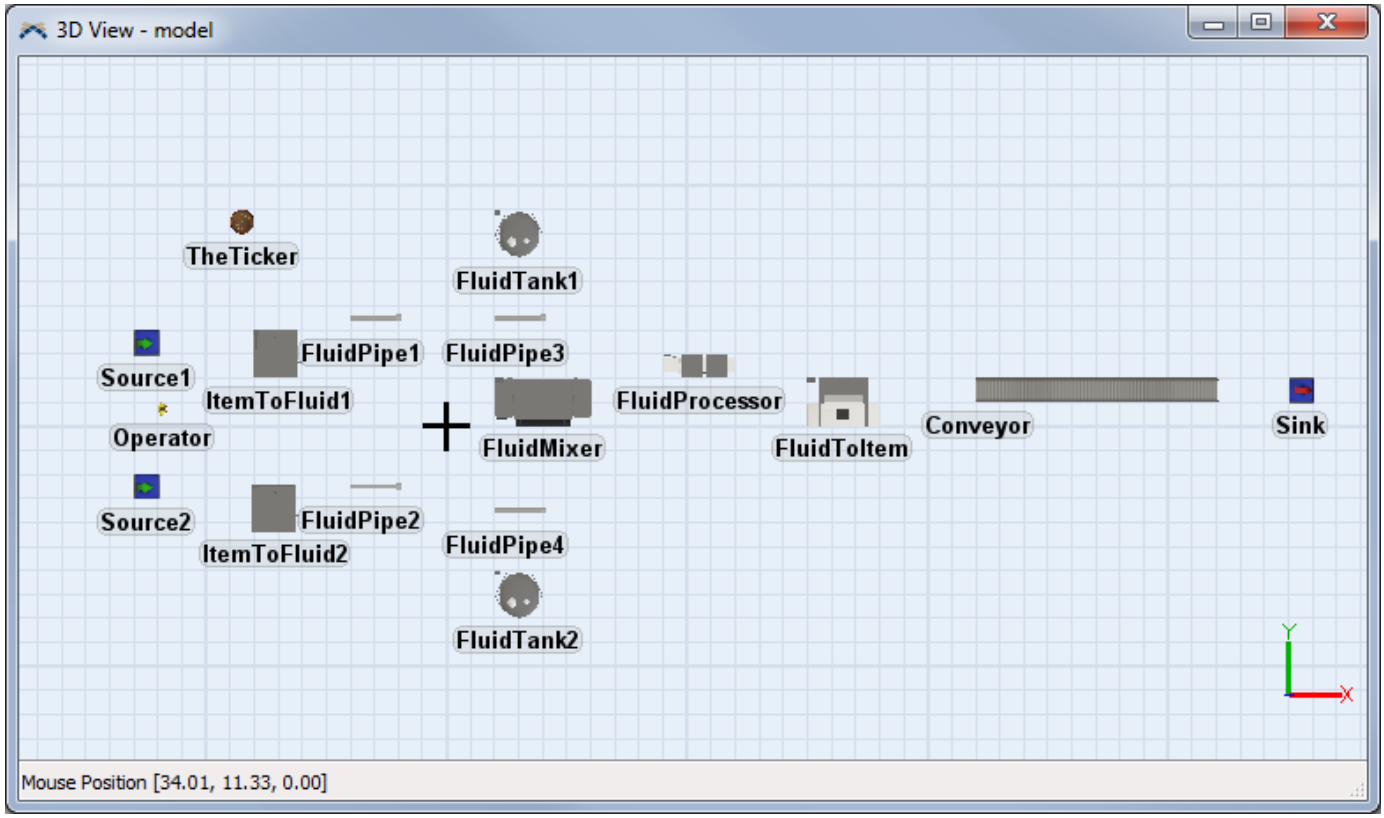
Step 1: Model Layout and Connections

To begin this lesson, drag out the following objects from the library. These objects can be found in the library by clicking on the plus button next to "Fluid" found near the bottom of the library window:



Note: When you create the first Fluid Object, a Ticker is automatically created and placed at (0,0). You can move this to any point in your model where it is not in the way, but do not delete it. It is required for the Fluid Objects to work.

- 2 **Sources** named *Source1* and *Source2* (Discrete Objects).
- 1 **Operator** named *Operator* (Discrete Objects).
- 2 **ItemToFluids** named *ItemToFluid1* and *ItemToFluid2* (Fluid Objects).
- 2 **FluidPipes** named *FluidPipe1* and *FluidPipe2* (Fluid Objects).
- 2 **FluidTanks** named *FluidTank1* and *FluidTank1* (Fluid Objects).
- 2 more **FluidPipes** named *FluidPipe3* and *FluidPipe4* (Fluid Objects).
- 1 **FluidMixer** named *FluidMixer* (Fluid Objects).
- 1 **FluidProcessor** named *FluidProcessor* (Fluid Objects).
- 1 **FluidToItem** named *FluidToItem* (Fluid Objects).
- 1 **Conveyor** named *Conveyor* (Discrete Objects).
- 1 **Sink** named *Sink* (Discrete Objects).
- Arrange the objects as shown below.

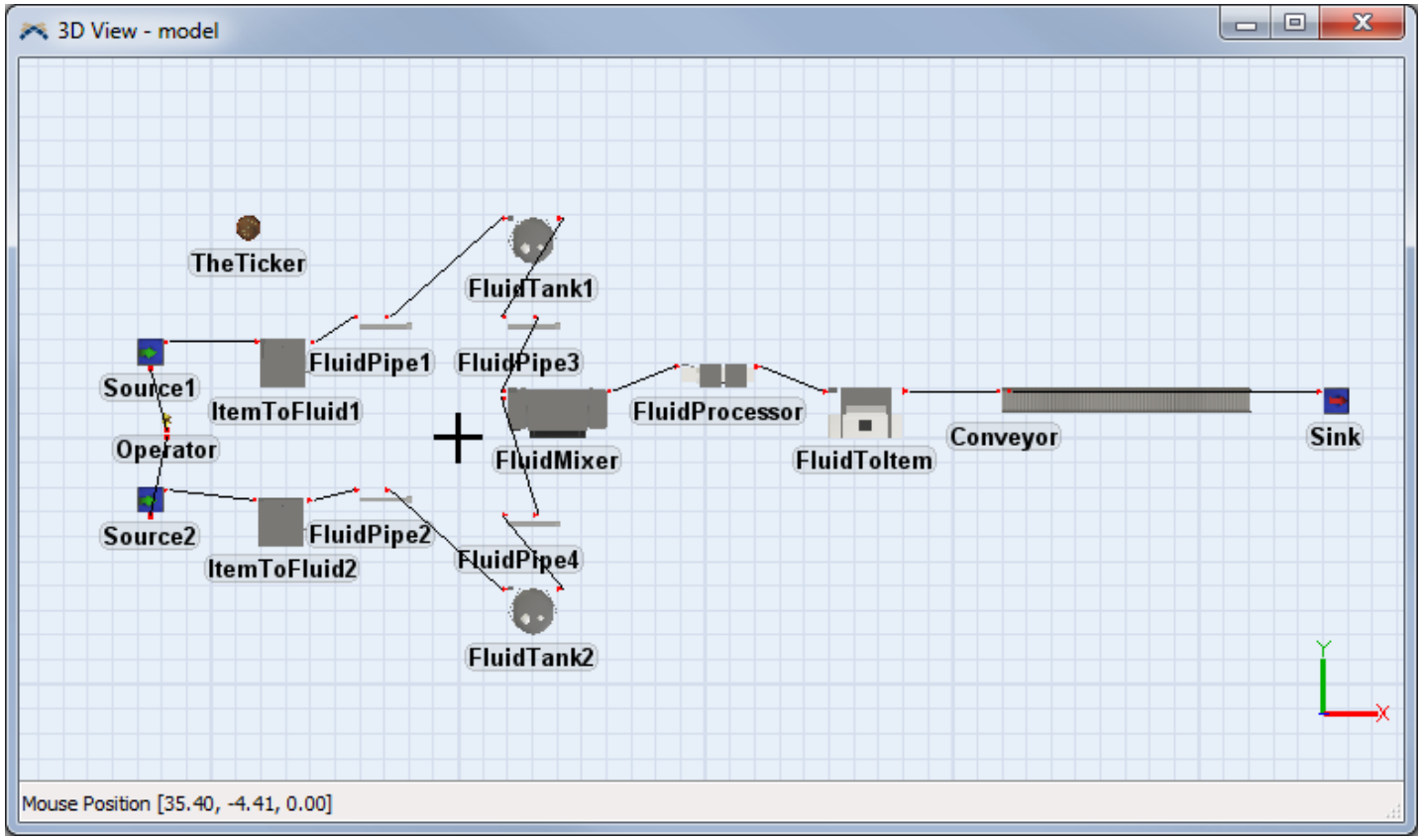


Once the objects have all been created and positioned where you want them in the model, they need to be connected. You use the same keys to connect Fluid Objects as you do to connect Discrete Objects: the A key creates an input/output connection and the S key creates a center port connection.

- Connect the objects so that there is a processing line
  - from *Source1* to *ItemToFluid1*
  - from *ItemToFluid1* to *FluidPipe1*
  - from *FluidPipe1* to *FluidTank1*
  - from *FluidTank1* to *FluidPipe3*
  - from *FluidPipe3* to *FluidMixer*
  - from *FluidMixer* to *FluidProcessor*
  - from *FluidProcessor* to *FluidToItem*
  - from *FluidToItem* to *Conveyor*
  - from *Conveyor* to *Sink*

  in that order, as shown below.

- Make a parallel line of connections from *Source2* to *FluidMixer*
- Both sources should call for the *Operator* to transport the flowitem to *ItemToFluid*, so a center port connection needs to be made from each *Source1* and *Source2* to the *Operator*.



Step 2: Configure the Sources

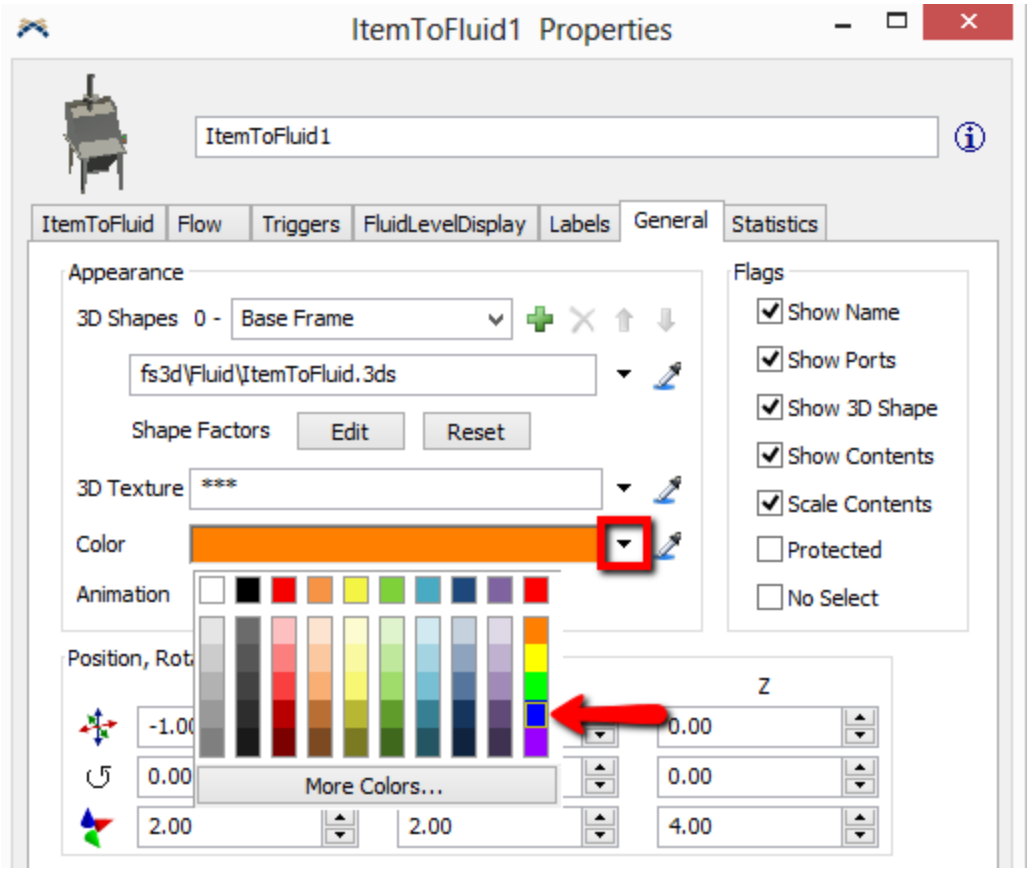
The default inter-arrival time for the Sources will work well for this model. They just need to call an Operator to transport the flowitems they create to the ItemToFluid objects.

- Click on *Source1* to open its properties in the Quick Properties. Under the **Flow** section check **Use Transport**.
- Repeat this step for *Source2*.

Step 3: Set the Colors of the Objects

When the objects are created they will be different colors, depending on their class. It is often helpful to color the objects based on the type of material that they will be processing. In this model there are two processing lines that are each composed of an ItemToFluid, a Pipe, a Tank and another Pipe. Using the Properties GUI, we will change the color of those four objects on one line to blue and the objects on the other line to red. This will cause pieces of the objects to change to that color as the model is running.

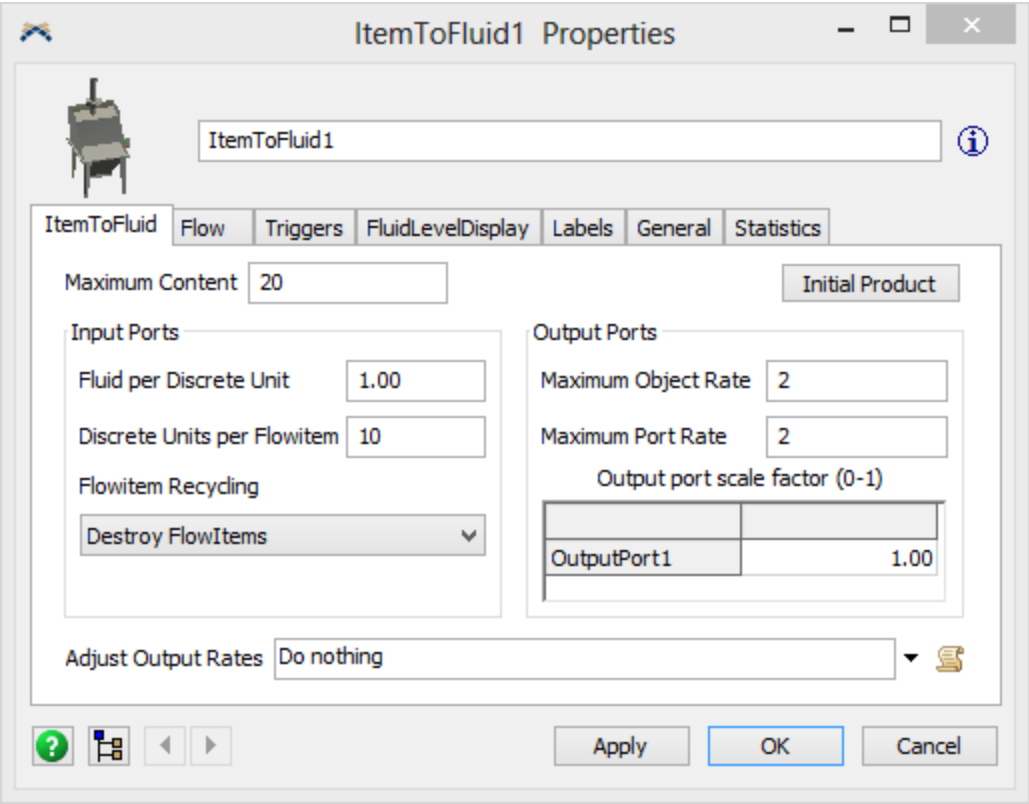
- Double-click on *ItemToFluid1* to open its **Properties** window, and click the **General** tab.
- Click the ▼ button located next to the **Color** field. The color popup will appear. Select a blue color from the the options, then click **OK**.
- Click **OK** to apply the changes and close the **Properties** window.
- Repeat this step for the rest of the objects mentioned above, coloring the other processing line red instead of blue.



Step 4: Configure the ItemToFluid

Next the ItemToFluid objects have to be configured to create the correct amount of material for each flowitem that comes in.

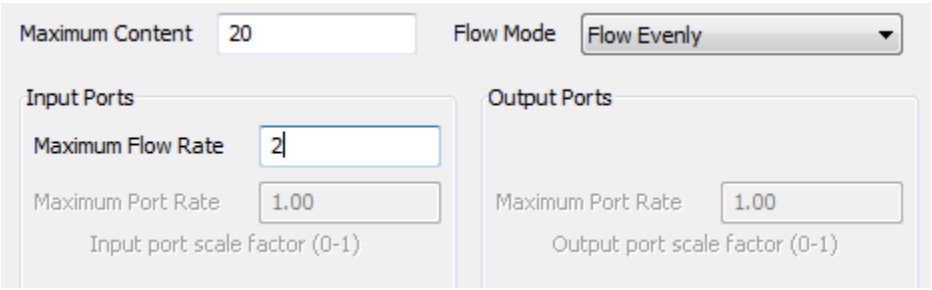
- Double-click *ItemToFluid1* to open its **Properties** window. On the **ItemToFluid** tab, change **Discrete Units per Flowitem** to 10. This tells *ItemToFluid* to create 10 gallons of fluid for each flowitem that enters.
- Change **Maximum Object Rate** and **Maximum Port Rate** to 2.
- Change the **Maximum Content** to 20.
- Click **OK** to apply the changes and close the **Properties** window.
- Repeat this step for *ItemToFluid2*.



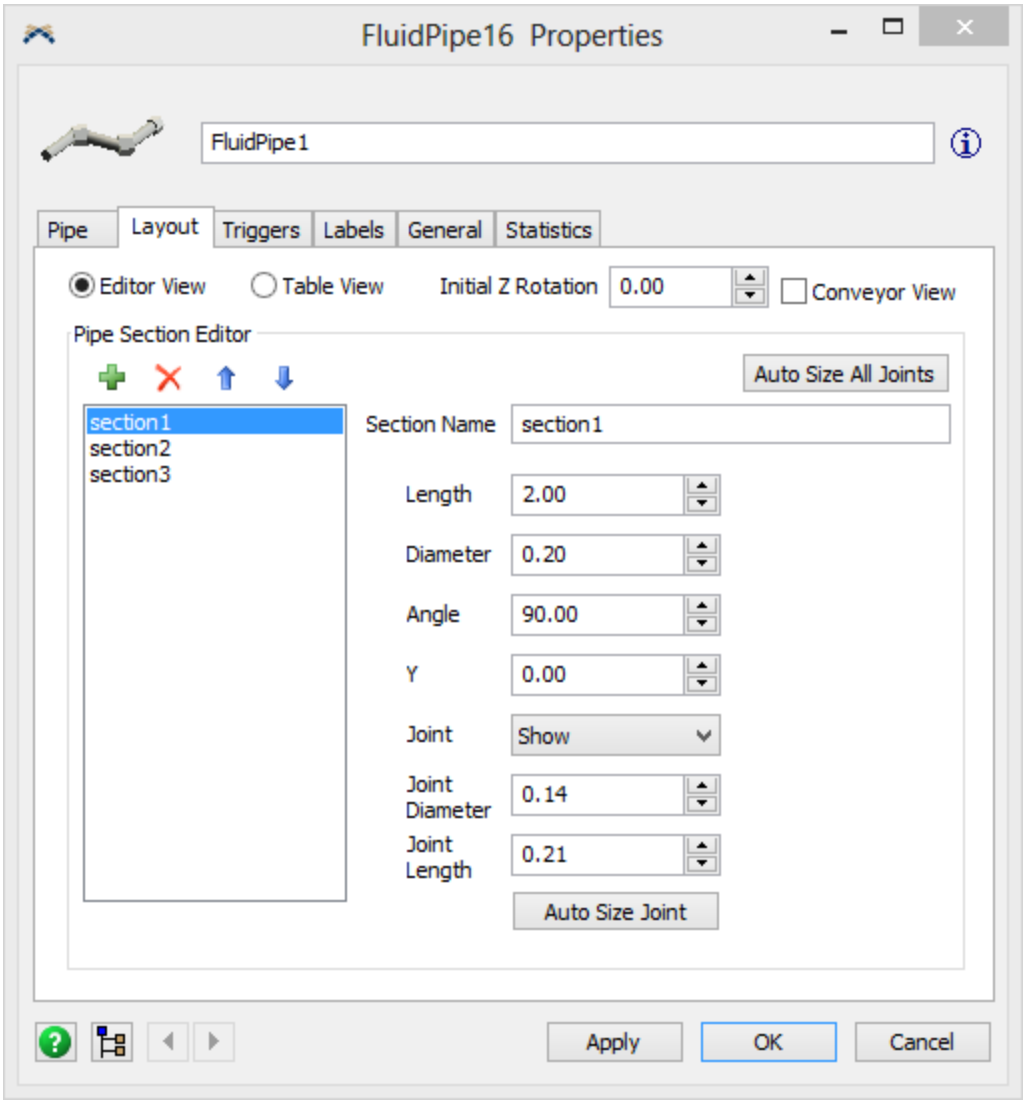
Step 5: Configure the FluidPipes

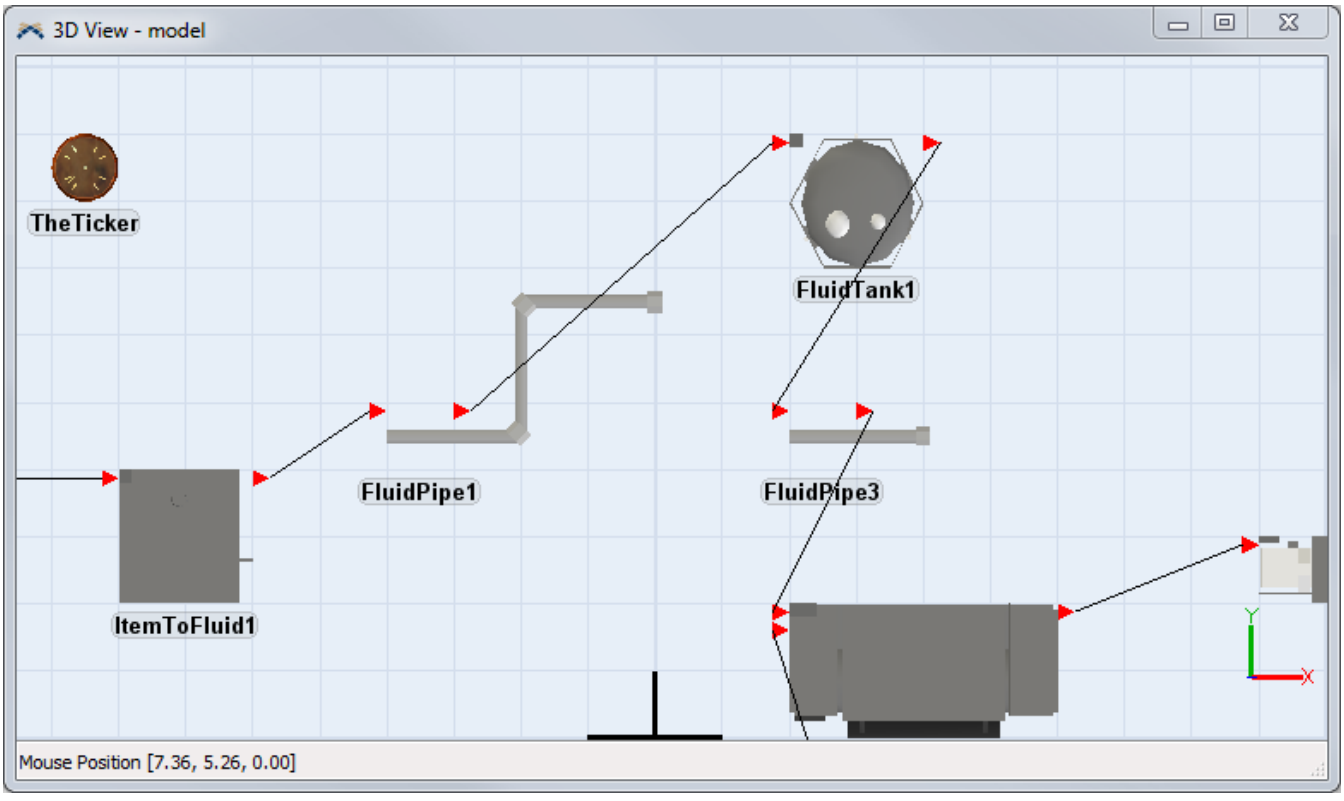
The Pipes that lead away from the ItemToFluids are the next objects that need to be configured. Pipes do not allow the modeler to specify the input and output rates. The output rate is based on the actual rate that material was received.

- Double-click *FluidPipe1* to open its **Properties** window.
- On the **Pipe** tab, set **Maximum Flow Rate** to 2, and set the **Maximum Content** to 20. This will ensure that material spends time in the Pipe, but not too much time.



- Click the **Layout** tab. Adjust the FluidPipe, changing and/or adding sections of pipe, so that it appears to start at *ItemToFluid* and end near *FluidTank* (the actual values of your FluidPipe may be different than the ones shown below). Changing the Layout does not affect the behavior of the pipes. More information about this tab can be found here.


FluidPipe16 Properties
FluidPipe1
Pipe
Layout
Triggers
Labels
General
Statistics
Editor View
Table View
Initial Z Rotation
0.00
Conveyor View
Pipe Section Editor
Auto Size All Joints
section1
section2
section3
Section Name
section1
Length
2.00
Diameter
0.20
Angle
90.00
Y
0.00
Joint
Show
Joint Diameter
0.14
Joint Length
0.21
Auto Size Joint
Apply
OK
Cancel

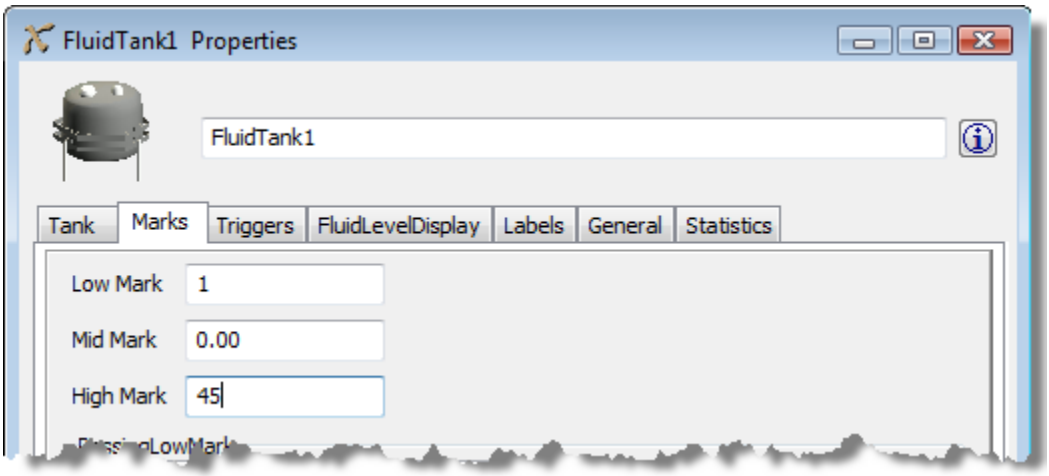

3D View - model
TheTicker
FluidTank1
FluidPipe1
FluidPipe3
ItemToFluid1
Mouse Position [7.36, 5.26, 0.00]

- Click the **OK** button to apply the changes and close the **Properties** window. Repeat this step for *FluidPipe2*.

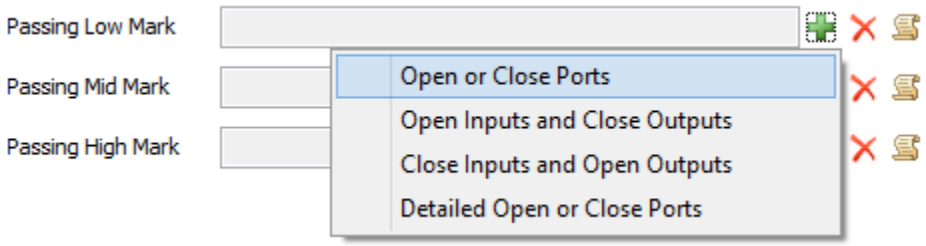
Step 6: Configure the FluidTanks

The ItemToFluids now have a maximum output rate of 2, but the FluidTanks have a maximum input rate 1. If these values are left as they are, the FluidTank's rate will be used during the model run (because it is the lower of two values) and the FluidPipe will not be able to send material downstream as fast as you have told it to. So the rate on the FluidTanks needs to be changed. FluidTanks allow the modeler to set three levels that will cause triggers to fire when the content of the FluidTank reaches them. These values are called marks. They are edited on the Marks tab of the Properties Window. For this model, the Tanks should keep their output ports closed until they have received a certain amount of material. They will then open the output ports and leave them open until the Tank becomes empty. They will always keep their input ports open.

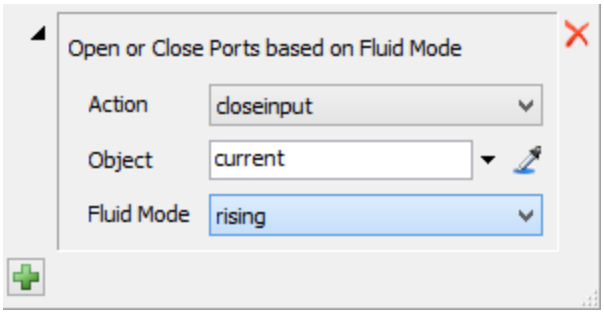
- Double-click *FluidTank1*. On the **Tank** tab, under **Input Ports**, change the **Maximum Object Rate** and the **Maximum Port Rate** to 2.
- Click the **Marks** tab. Change the **Low Mark** to 1, and the **High Mark** to 45. Leave the **Mid Mark** at 0. If a mark has the value 0, the trigger for that mark will never fire.



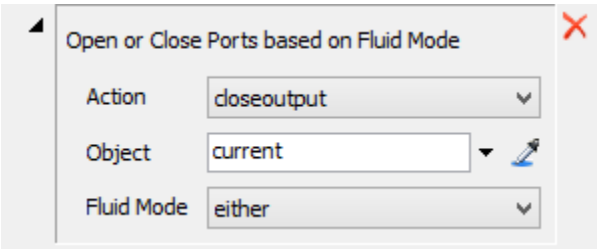
- On the **Marks** tab, click the + to add a function to the **Passing Low Mark** trigger. Select the **Open or Close Ports** option.


Passing Low Mark
Passing Mid Mark
Passing High Mark
Open or Close Ports
Open Inputs and Close Outputs
Close Inputs and Open Outputs
Detailed Open or Close Ports

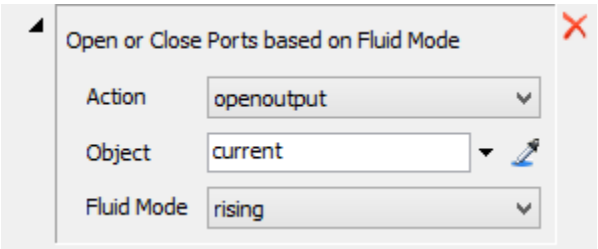
- Click where it says rising. The available options will appear.



- Give the function the following parameters:



- Add the **Open or Close Ports** function to the **Passing High Mark** trigger in the same way. Give it the following parameters:



- Click **OK** to apply the changes and close the **Properties** window.
- Repeat the same procedure for **FluidTank2**.

Step 7: Reorient and resize the next Pipes

Depending on where you placed the objects in your model, the Pipes that lead from the Tanks to the Mixer may need to be adjusted so that they point toward the Mixer. Again, this is completely visual, the size and layout of the Pipes will not affect their behavior. Use the Layout tab to configure the Pipes in your model so that they look good to you. The default maximum content of these Pipes is currently too large for this model.

- Double-click *FluidPipe3* to open its **Properties** window. On the **Pipe** tab, change the **Maximum Content** to 10. This will make sure that the material leaving the Tanks takes just a little time to get to the Mixer.
- Repeat this step for *FluidPipe4*.

Step 8: Set up the Mixer's step and recipe tables

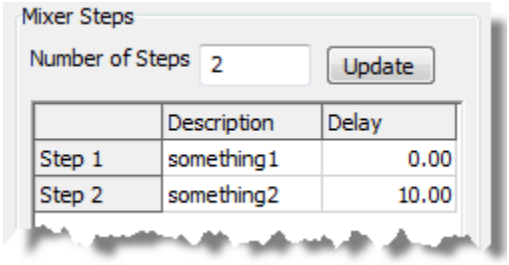
The FluidMixer now needs to be configured to receive the two different materials and combine them into a new material. This is done by changing the two tables found on the Steps tab in the FluidMixer's Properties window. The Mixer Steps table is used to define a series of steps that the Mixer must go through for each

batch that it processes. In this model, the Mixer Steps table needs 2 steps. The Mixer should pull 10 gallons of the first material from input port 1 during step 1. Then it should pull 20 gallons of the second material from input port 2.

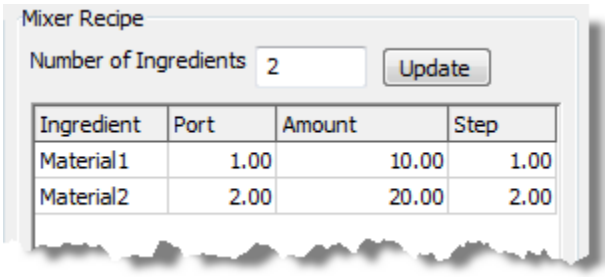
- Double-click the *FluidMixer* to open its **Properties** window, and click the **Steps** tab.
- Under **Mixer Steps**, change the **Number of Steps** to 2.
- Click **Update**. The steps will appear in the table.

The description of the steps is not important, call them anything you want. The delay time for each step is executed after all the material for that step is received, and before the Mixer starts receiving material for the next step.

- Set the **Delay** for **Step 1** to 0, and the **Delay** for **Step 2** to 10.

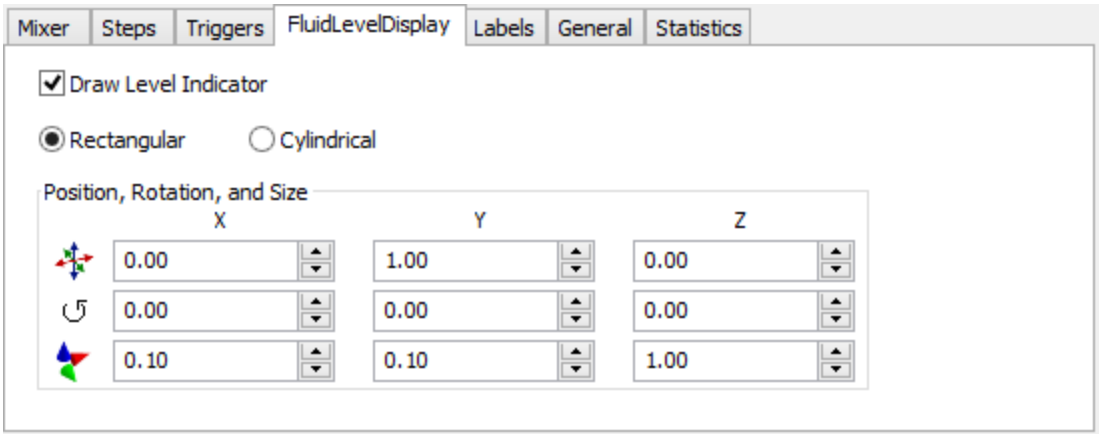


- On the **Steps** tab, under **Mixer Recipe**, change the **Number of Ingredients** to 2.
- Click Update. The ingredients will now appear in the table. Once more, the description can be anything you want because the object does not refer to these names. Call them *Material1* and *Material2*.
- For *Material1*, change the **Port** to 1, the **Amount to** 10, and the **Step** to 1.
- For *Material2*, change the **Port** to 2, the **Amount** to 20, and the **Step** to 2.



The Mixer's level display is a useful tool to watch during the model run. It will display how much of each material in the Recipe Table the Mixer has received at any point in time. By default it is hidden behind the Mixer's 3D shape. More information about the fluid level display is found here. You can also manipulate the size and shape of the fluid level display so that it looks like part of the object to get a more realistic look.

- Click the FluidLevelDisplay tab. Change Y to 1.



Step 9: Examine the FluidProcessor

The FluidProcessor's default values will work well for this model. It will receive material from input port 1, process it for a certain amount of time and send it to output port 1. The amount of time it spends processing is based on its **Maximum Content** and the **Maximum Output Rate** that the modeler defines in the FluidProcessor Properties window on the **FluidProcessor** tab. You can play with these values if you want to see how they affect the model. To make the bar on this object appear in front like on that of the *Mixer*, you will once again need to change the Y location of the level indicator bar to 1, as done in the previous step.

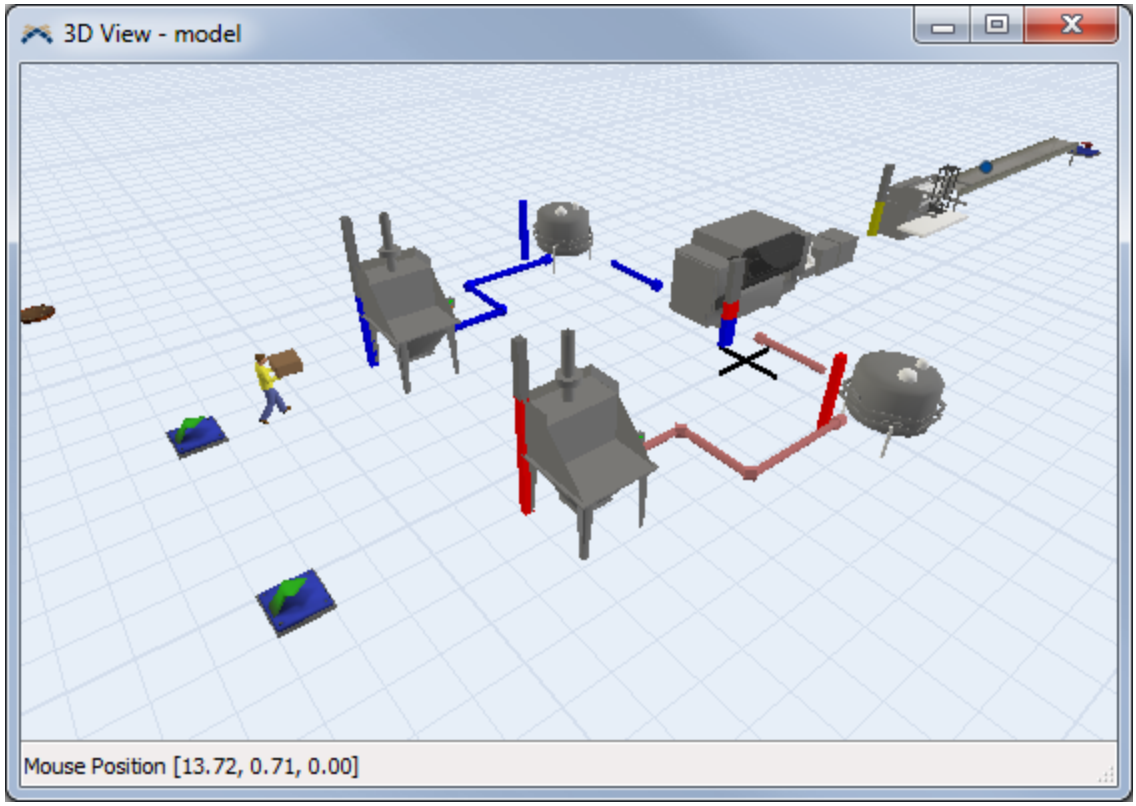
Step 10: Configure the FluidToItem

The FluidToItem near the end of the line will convert the fluid material coming from the FluidProcessor and change it into flowitems. The Fluid per Discrete Unit and Discrete Units per Flowitem will be multiplied together to determine how much material will have to be collected to create a single flowitem. In this case, 10 gallons of fluid will become 1 flowitem.

- Double-click the *FluidToItem* to open its **Properties** window.
- On the **FluidToItem** tab, under **Flowitem Output**, change **Fluid per Discrete Unit** to 10.
- On the **FluidToItem** tab, change the **Maximum Content** to 10. This tells the object that it can only collect enough material for 1 flowitem at any time. If this value is higher, the *FluidToItem* may end up acting as a Queue and creating too much storage space in your model.

Step 11: Reset and run the model.

- **Reset** and **Save** the model.
- **Run** the model. You should see the level indicator bars on the objects going up and down. You should also see the Pipes flashing. If a Pipe is drawn in gray, it is empty. If it is a pulsing color, material is flowing through the pipe. If it is a solid color, the material is blocked. There will also be flowitems being changed into fluid and fluid being changed into flowitems.



After completing this lesson your should have an idea of how these Fluid Objects work and some of their capabilities. There is far more that they can do than was covered in this lesson. Read through their documentation and try other options and settings. Soon you'll be putting together larger, more detailed fluid-based models.

Modeling Views

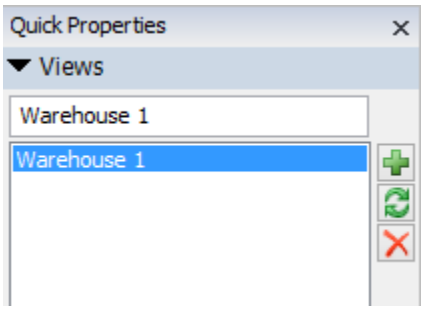
1. Orthographic/Perspective View
2. Tree Window
3. Travel Networks
4. Modeling Utilities
    - Library Icon Grid
    - View Settings
    - Light Source Editor
    - Quick Properties
    - Edit Selected Objects
    - Find Objects
    - Groups
    - Model Layouts
    - Measure / Convert
    - Command Helper
    - Keyboard Layout

Orthographic/Perspective View

Topics

- **Saving Views**
- **View Settings**
- **Moving Around in the View**
- **Moving Objects**
- **Sizing Objects**
- **Connecting Objects**
- **Creating and Editing Selection Sets**
- **Editing an Object's Properties**
- **Right-Click Menu**

Saving Views



The Views utility is found in the Quick Properties window when there are no objects selected in the 3D view. It allows you to capture the view's current camera position and add it to a list of views.

View Name - Rename the selected view by typing a new name in this field.

View List - Select a view from the list to rotate/move to that view.

- Captures the current view.

- Updates the currently selected view to the camera's current position/rotation.

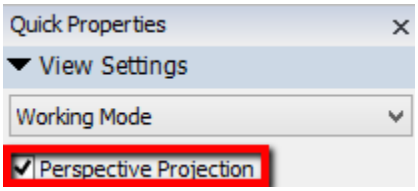
- Removes the selected view from the list.

Views can also be access through the right-click menu.

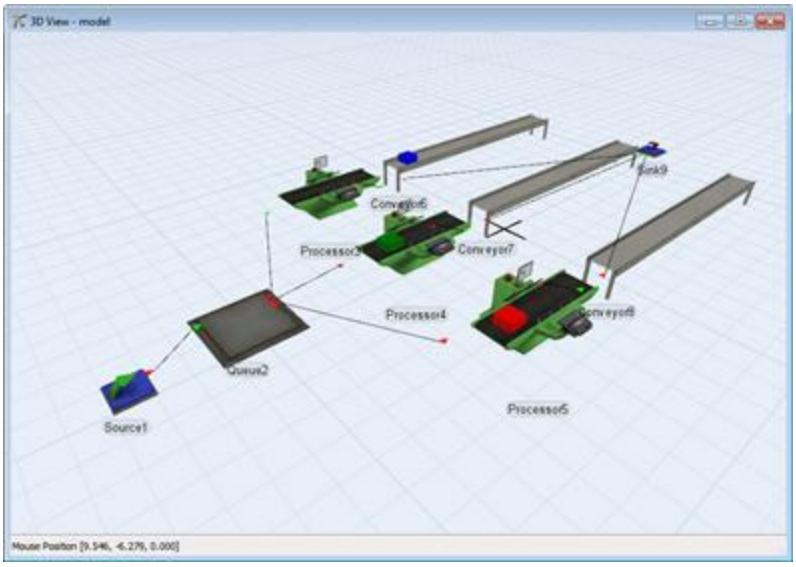
View Settings

For more available view settings, see the View Settings.

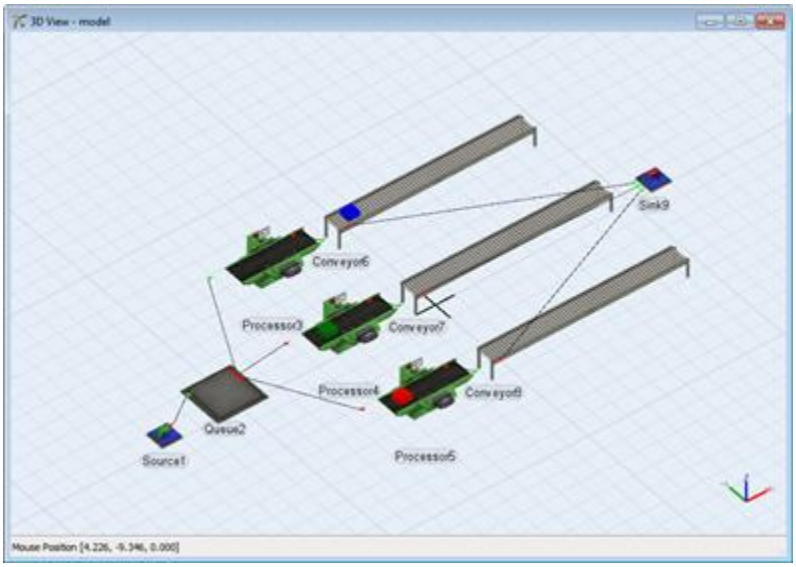
The 3D view may be toggled between orthographic and perspective modes by deselecting all objects in the 3D view and checking/unchecking the **Perspective Projection** option in the Quick Properties window.



Perspective projection gives the model a more 3D world feel.



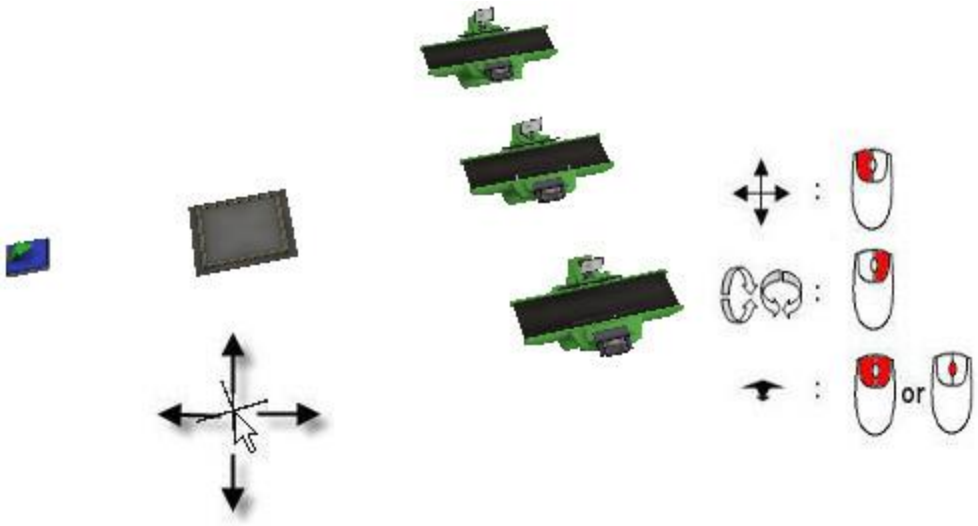
Where as the orthographic view has no depth.



For more information about changing the View Settings for the 3D View, see the View Settings page.

Moving Around in the View

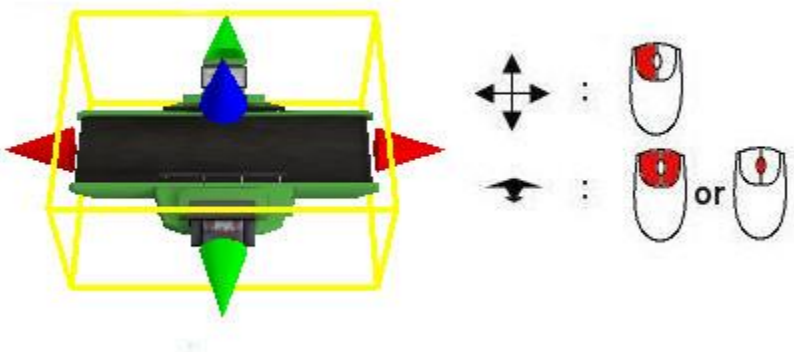
To move around in the view, click-and-hold on the floor of the model, and drag the mouse around in the view. This will translate the camera. To rotate the view, right click-and-hold on the floor of the model and drag the mouse in different directions. To zoom in and out, hold down both the right and left mouse buttons and move the mouse up and down. You can also zoom in and out by scrolling the mouse wheel.



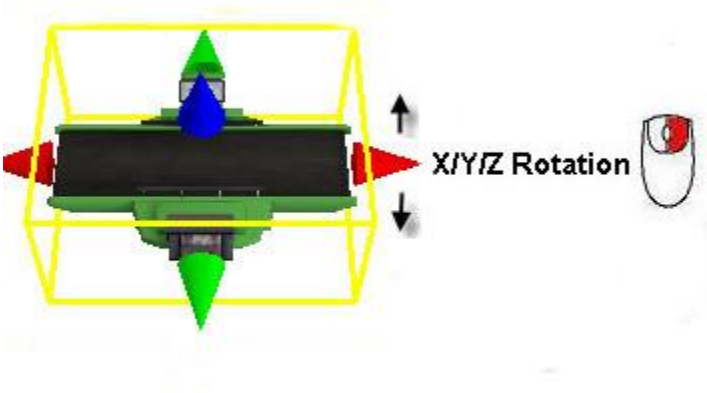
If you are working in a perspective view, you can also do a mouse guided fly through using the F8 key. Make sure the mouse is in the center of the window. Then press the F8 key. Move the mouse up and down to fly forward and backward. Move the mouse left and right to turn left and right. Once you are finished, click on the F8 button again to exit fly-through mode. It is sometimes easier to navigate if the view is configured as first person (from the view settings window).

Moving Objects

To move an object in the X/Y plane, click-and-hold on the object and drag it to the desired location. To move the object in the z direction, click on it and scroll the mouse wheel. You can also hold both the left and right mouse buttons down on the object and drag it up and down.



To rotate the object, click on one of the three axis arrows with the right mouse button and drag the mouse up and down.



Sizing Objects

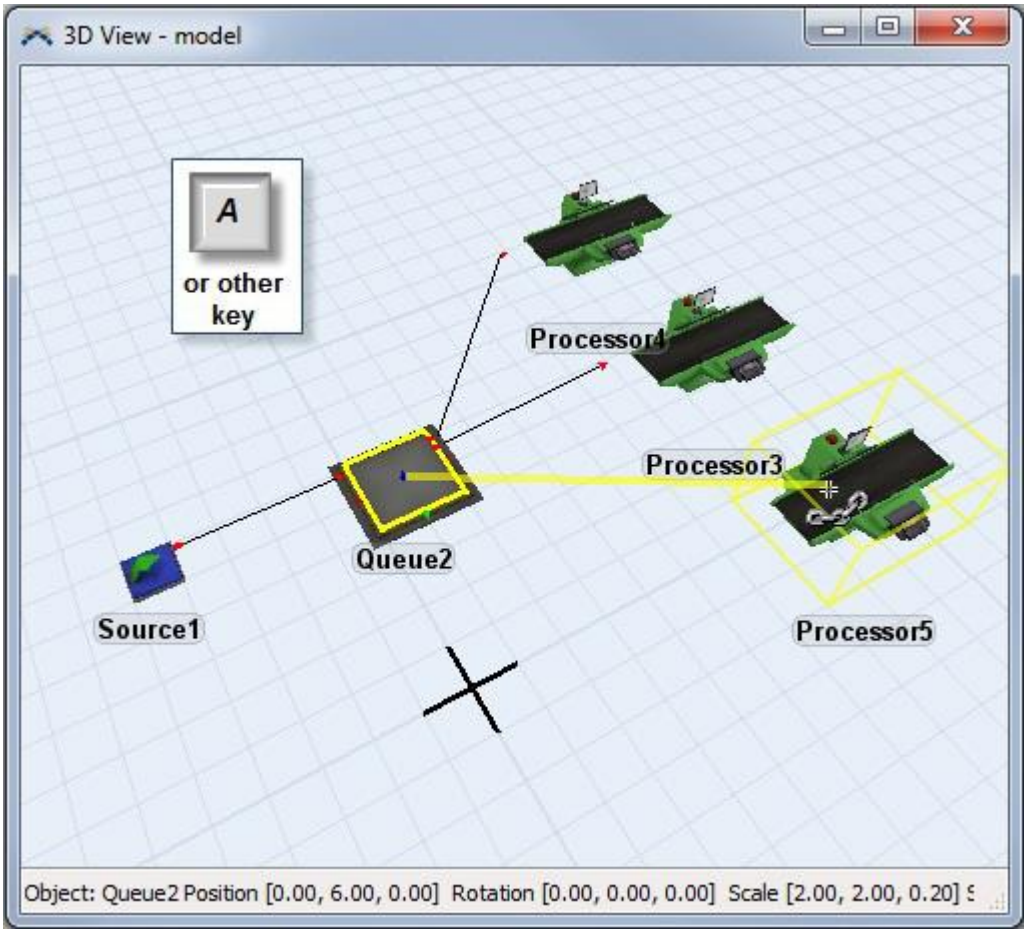
To change the object's size, click on one of the three axis arrows with the left mouse button and drag the mouse in the direction of the arrow. You can size the object in all three axes by 5% by pressing Ctrl and "L" to size up or "K" to size down. You can also size the objects in all three axes at once by holding alt and clicking on one of the three axes and moving the mouse in the direction of the axis. If the three axis arrows are not showing, go to the Edit menu and select Resize and Rotate Objects.

Connecting Objects

To connect two objects in the model, hold the 'A' key down, click-and-hold on one object, drag the mouse to the other object, and release the mouse button on that object. The 'A' connect method usually connects output ports to input ports, but you can also use other key connections. For instance, the 'Q' key is used for disconnecting. These and others are described in detail in the keyboard interaction section.

Another way to connect objects is to use the Connect Objects mode on the main toolbar. Having this mode selected allows you to create connectionsa without holding the 'A' key.

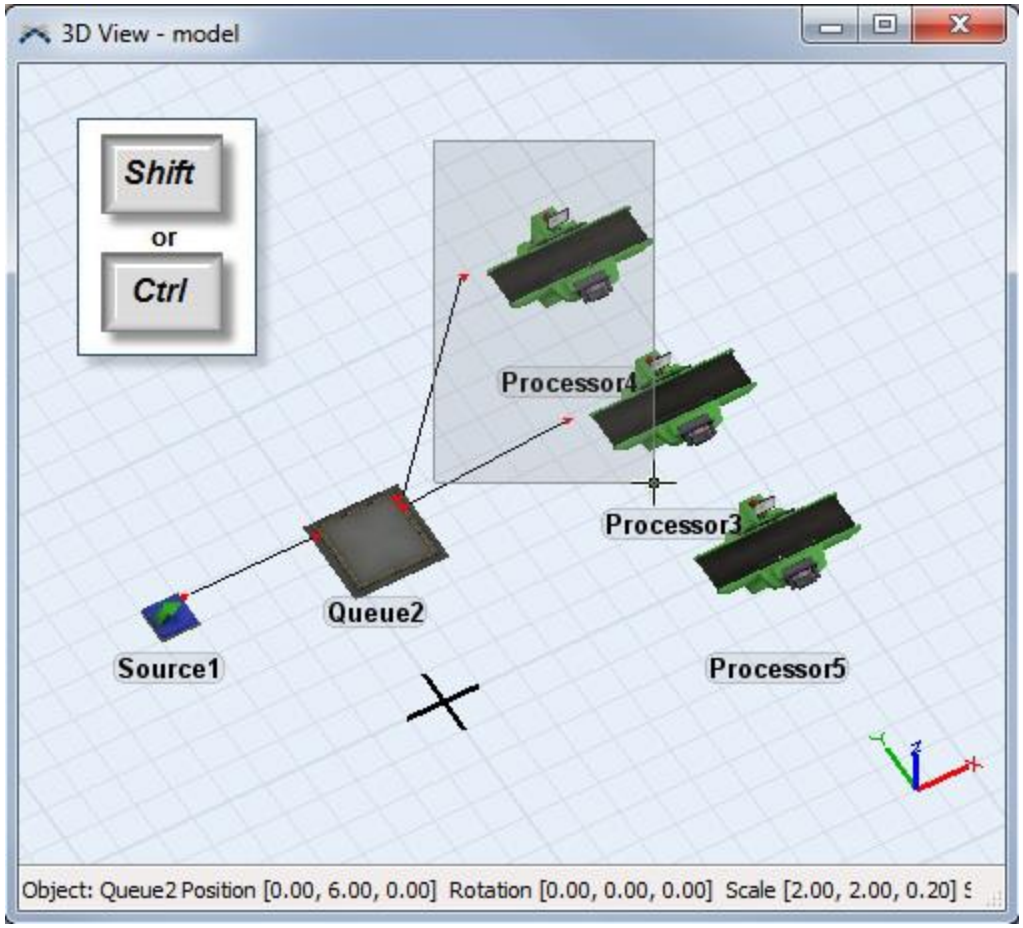
You can also create multiple connections in series. To do this, you hold 'A' or use Connect Objects mode , click an object, then click another object, then click another object, and so on.



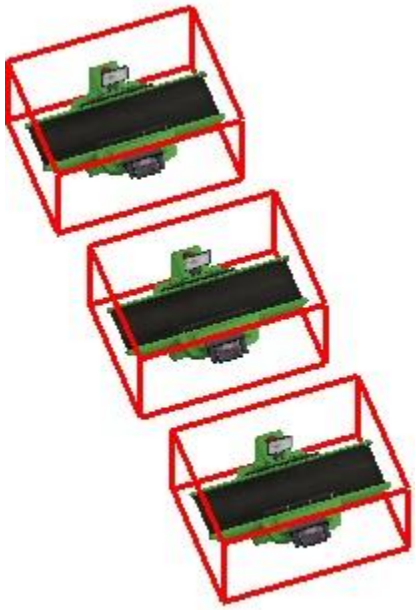
Creating and Editing Selection Sets

You can create selection sets to have operations apply to a whole set of objects. To add objects to the set, hold the Shift or Ctrl key down and drag a box around the objects that you want selected. Holding Shift down resets the selection set, while holding Ctrl down adds or removes the objects to the selection set. You can also hold Shift or Ctrl down and click on objects, instead of dragging a box around them.

Another way to create selection sets is to use the New Selection mode on the main toolbar. This mode works the same as holding the Shift key down. The Toggle Selection mode works the same as holding the Ctrl key down.



Objects in the selection set are drawn with a red wire frame around them.



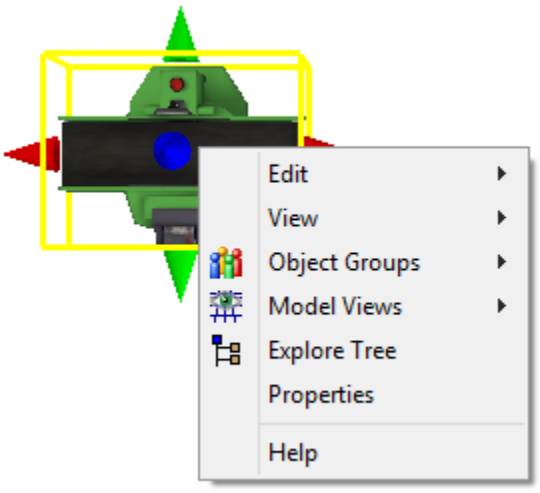
Once you have created a selection set, moving, rotating and scaling one of the objects will cause the other objects in the selection set to be moved, rotated or scaled as well. You can also perform several operations on the selection set from the Edit Selected Objects menu.

Editing an Object's Properties

To edit the properties of an object, refer to the Quick Properties window on the right side of your screen, or double click on the object or right click on it and select "Properties" from the popup menu to bring up more properties.

Right-Click Menu

Right click on the Perspective View or on an object to display this popup menu:

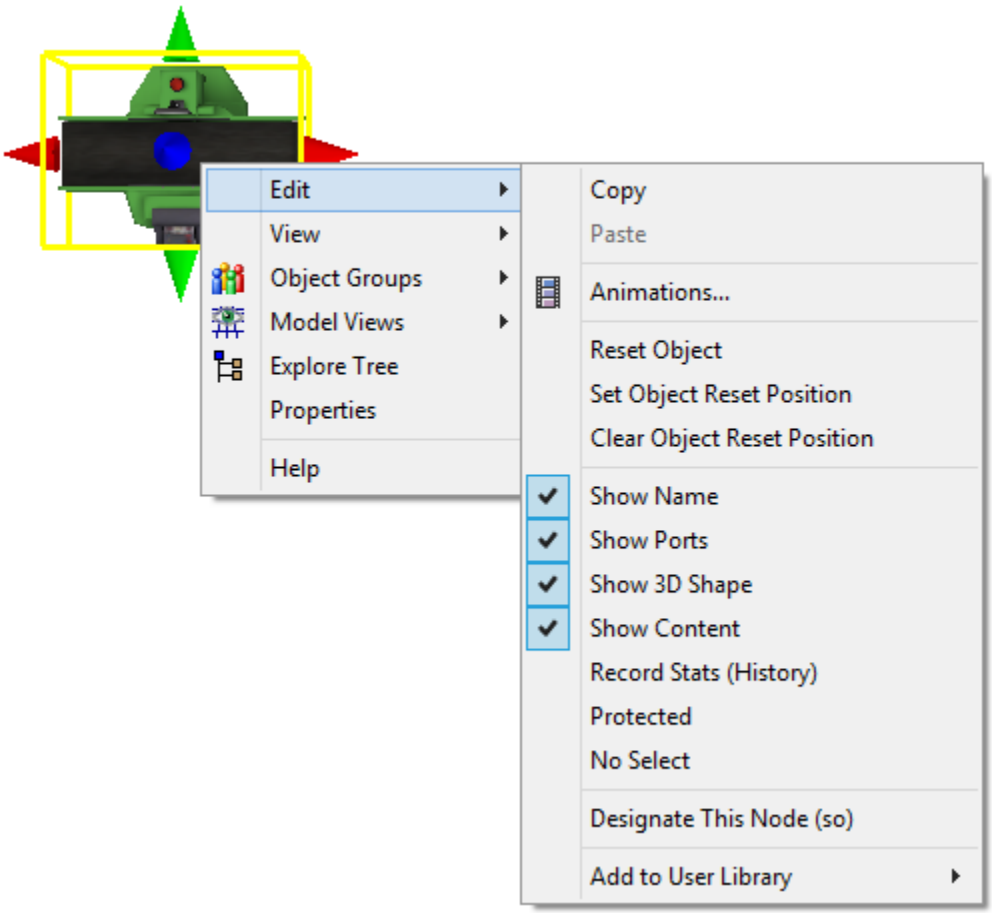


Explore Tree - Opens a Tree Window and displays the object in the Tree.

Properties - Opens the object's properties window.

Help - Displays this help page.

Edit



Copy - This copies the current item(s) to the clipboard.

Paste - This pastes from the clipboard into the current item (usually the model).

Animations... - Opens the Animation Creator to edit the object's animations.

Reset Object - This resets the x/y/z rotation and the z location of the object to 0.

Set Object Reset Position - This saves the object's current position so that whenever you press the reset button, the object will go back to that position (particularly useful for mobile objects such as transporters and operators).

Clear Object Reset Position - Clears the object's Reset Position.

Show Name - When this option is selected, the object's display name is always visible.

Show Ports - This toggles visibility of the ports and connections of the item.

Show 3D Shape - This toggles object visiblity in 3D views.

Show Content - This toggles flowitem visibililty within the object (i.e., flowitems in a Queue).

Record Stats (History) - This is another way to turn on statistics recording for the object.

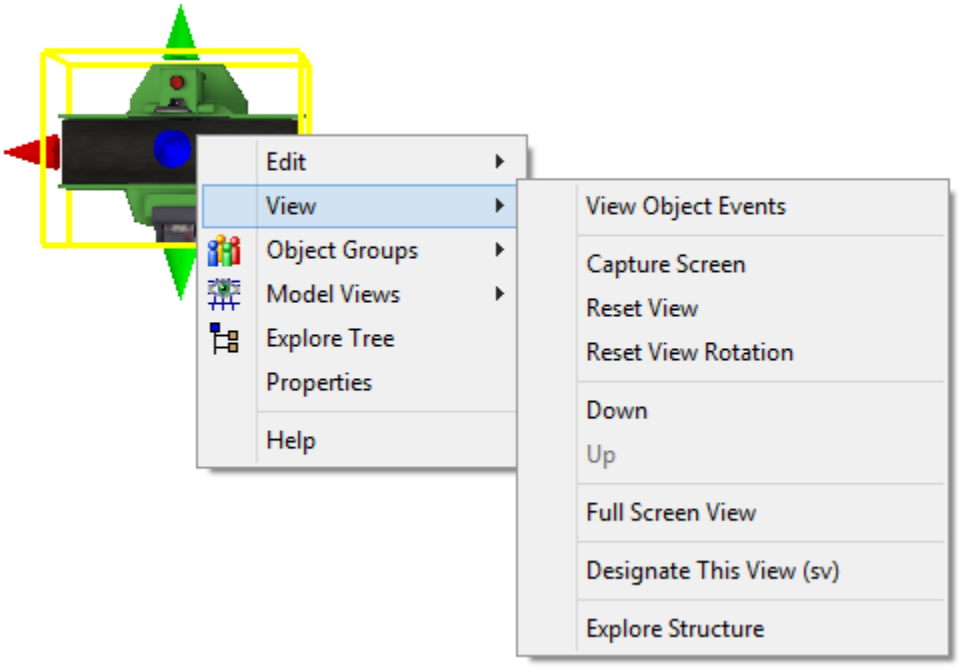
Protected - This locks the objects' location, size and rotation.

No Select - This toggles the object's No Select, making the objects unclickable in the 3D View (still accessible in the Tree Window).

Designate This Node (so) - This designates the object as the "selected object", which can then be referenced in code as so(). You will usually use this option for writing code in the script console. There can only be one so() at any time.

Add to User Library - This option adds the selected object to the a selected user library.

View



View Object Events - Opens the Event List and displays all the current events for that object.

Capture Screen - This option captures the current screen, saving it as a bitmap file in the prints folder of flexsim. This can also be done by selecting on the active view and pressing the 'P' key.

Reset View - This returns the camera position of the view to its initial position, which is focused at 0,0,0.

Reset View Rotation - This returns the camera rotation of the view to its initial position, which is looking straight down.

Down - This option causes the view to drill down into the selected object, allowing you to see what is going on in the "contents" of that object. The title bar of the window will show the name of the object that you are observing. This feature is particularly useful for Visual Tools that hold other objects.

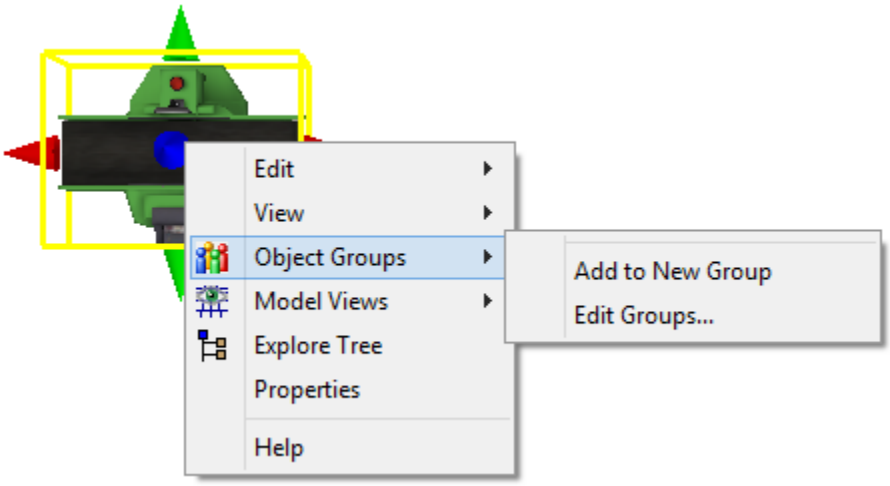
Up - This option causes the view to return up out of a selected object. If you have drilled down multiple levels you will have to return up the same number of levels to reach the main view.

Full Screen View - This sizes the window to take up the full screen. The window's title bar and frame will not be shown, and any other Flexsim windows on that monitor, including the main window, will be hidden behind this view. To exit Full Screen View, right click in the view and select View > Out of Full Screen View.

Designate This View (sv) - This designates the window as the "selected view", which can then be referenced in code as sv(). You will usually use this option for writing code in the script console or for use with the AVI Maker. There can only be one sv() at any time.

Explore Structure - This option brings up a tree window exploring the tree structure of the orthographic window itself.

Object Groups



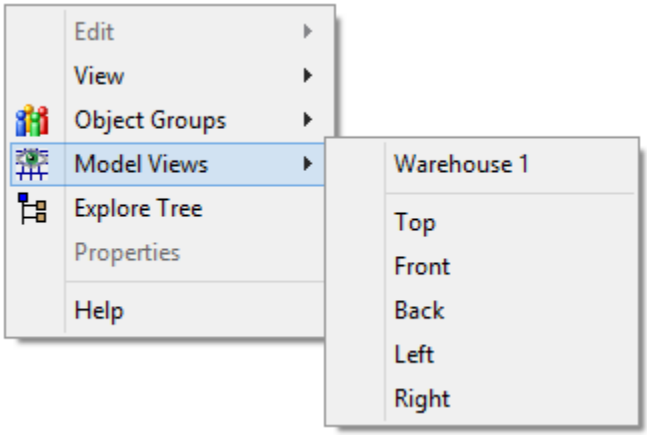
This list will dynamically update as you add groups. Each group name will be listed, and will become checkable, so you can easily add objects to any number of groups.

Add to New Group - This adds the object to a new group.

Edit Groups... - This opens the Object Groups utility.

For more information see the Groups page.

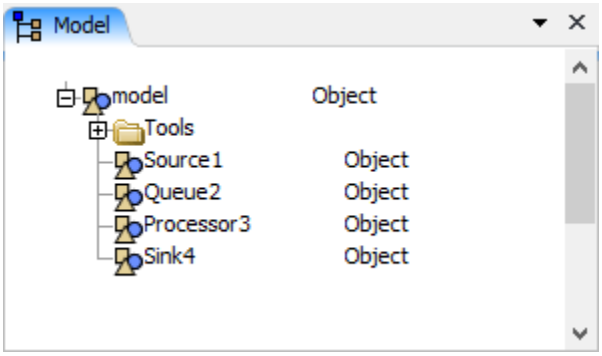
Model Views



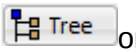
This list dynamically updates to show all available views. The defaults are **Top**, **Front**, **Back**, **Left**, and **Right**. Selecting an available view will rotate the model to that view.

Custom views, the ones created through the Views utility, will appear at the top of the menu.

Tree Window



For information about the structure of the tree, see the Tree Structure page.

The Tree Window can be accessed from the Toolbar by clicking the 

or from the View menu.

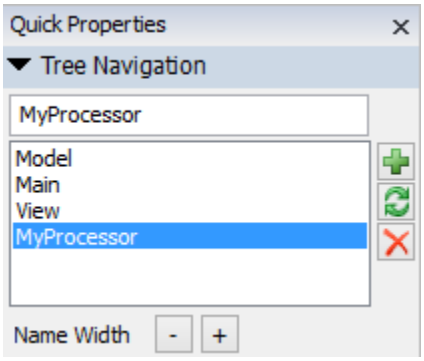
Navigating the Tree

To move around in the tree window, click the mouse on a blank area of the tree view and drag it around. You can also use the mouse wheel and page up/page down keys to scroll up and down in the tree window. You can also use the Tree Navigation panel from the Quick Properties, as described below, to jump around the Tree.

Quick Properties

When the tree window is active, the Quick Properties will change to display the Tree Navigation panel and the Search panel. If a node is clicked on in the tree that does not have object data, the Node Properties panel will also be displayed in the Quick Properties. Clicking on nodes with Object data will display similar results as clicking on them in the 3D view.

Tree Navigation



This panel allows you to navigate between sections in the FlexSim tree.

- **Model -** A subset of the Main Tree, this contains all of the objects used in the currently open model.
- **Main -** The Main Tree contains many of the higher level functions in FlexSim.
- **View -** The View Tree contains all of the GUIs in the FlexSim interface.

You also have the option of saving views from the Tree. By highlighting a node in the Tree and pressing the that node will be added to the list of preset views. This allows you to quickly jump back and forth between different sections of the Tree. Select a view and click the to remove it from the list. will update the selected preset view to the highlighted node.

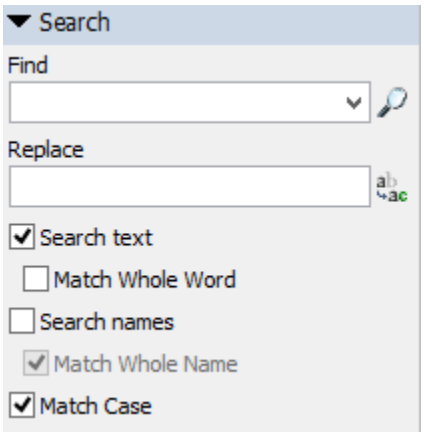
Rename the preset view by entering a new name in the name field.

Views that are saved are persistent even after FlexSim closes. Views that are added from the Model get saved into the Tools/TreeNavigation folder and wil be available to anyone who loads that model. Views that

are added from anywhere else in the Main Tree or from the View Tree are saved in the User Prefs and will be available whenever FlexSim is open under your user.

Name Width - This controls the name width of nodes in the Tree.

Search



The Search allows you to search through text and node names in the Tree. The Search will begin at the highlighted node in the active Tree Window. If no node is highlighted, the search will begin at the top of the active Tree Window. The search recursively searches through all subnodes and object attribute nodes.

Replace allows you to replace all occurrences of the found text. This can be applied to text and/or node names.

- Finds the specified text (or press the Enter key in the field).

- Finds the above text and replaces it with the specified text (or press the Enter key in the field).

Search text - Searches through all text under the selected node.

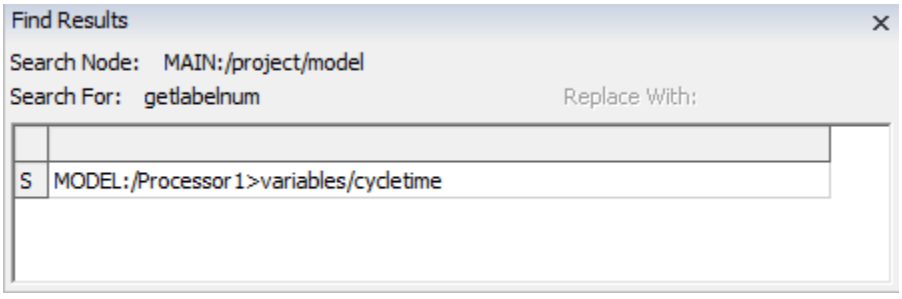
Match Whole Word - If checked, only finds whole word matches of the text (meaning characters immediately before and after the search text must be non-alphabetic characters).

Search names - Searches through all node names under the selected node.

Match Whole Name - If unchecked, the search will return any node name that contains the searched text.

Match Case - If unchecked, the search will find all text/names containing the search text, regardless of capitalization. For example, if you search for "myvariablename", the results will still return all nodes containing "MyVariableName".

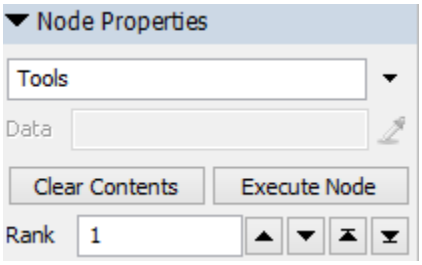
Once a search is begun, the Find Results window will appear:



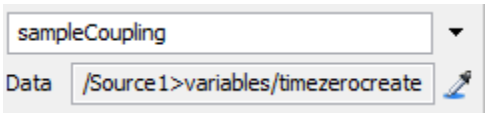
Double-click on a result to view the text in a Code Editor window. You may also right click the result to Edit Code or Explore Tree.

Nodes toggled as FlexScript will have an S to the left of the node path. Nodes toggled as C++ will display a C and nodes toggled as DLL linked will display a D.

Node Properties



When a non-object node is highlighted in the Tree, the Node Properties panel will appear,displaying the name and rank of the node. If the node has coupling data, the Data field will display the coupling's associated node. Use the to select the node to create a coupling with.



- This will display a menu of possible object attribute names. This is a shorted and organized list taken from the Attribute Hints window.

Clear Contents - Destroys all subnodes of the highlighted node.

Execute Node - Calls nodefunction or executefsnode on the node, executing its associated FlexScript, C++ or DLL code.

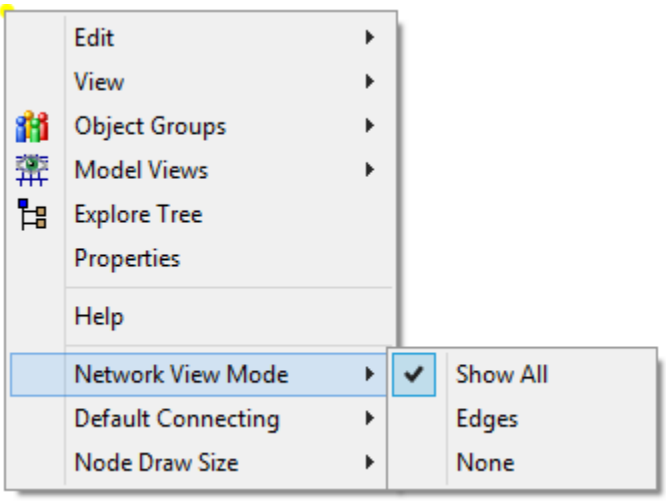
Rank - This specifies the rank of the highlighted node in its parent's tree. Enter a new rank number or use the up and down arrows to rerank the node.

Travel Networks Utility

The Travel Networks Utility has been moved since Version 6.0.2 and can now be found by dragging a Network Node into your model and right clicking on it.

Network View Mode

To change the way NetworkNodes are viewed in the 3D view, right-click on a NetworkNode and choose Network View Mode:



Alternatively, you can hold the X key down and left click on a NetworkNode to toggle between these three modes.

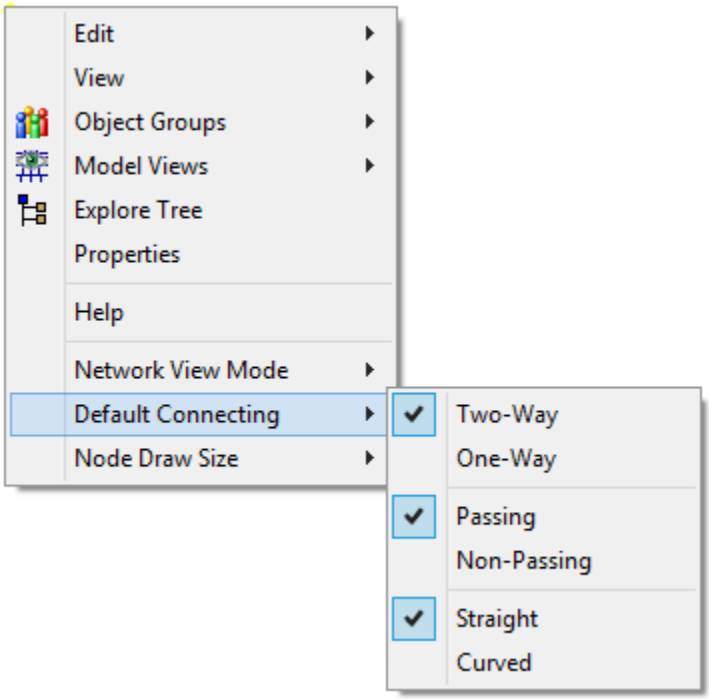
Show All - Displays the NetworkNodes, the edges between them, and the direction arrows.

Edges - Displays the edges between NetworkNodes only.

None - Hides all NetworkNodes and edges except for the NetworkNode that was right-clicked on.

Default Connecting

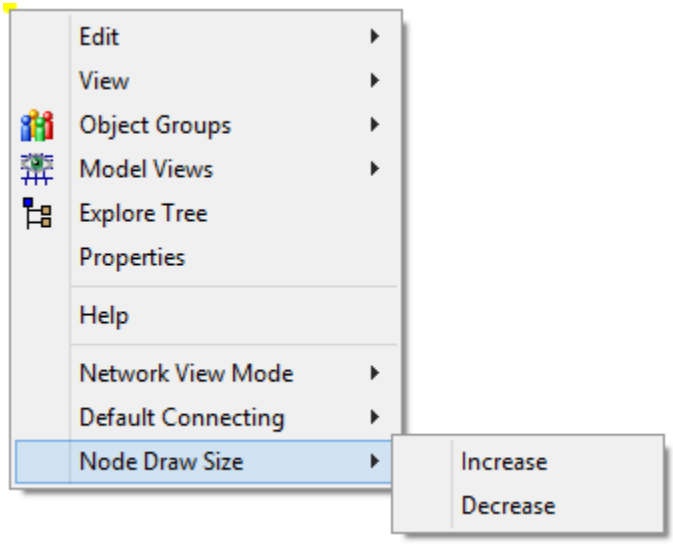
Default Connecting specifies how new connections will be made between NetworkNodes when you do an 'A' drag. These parameters may be changed on each individual connection after it has been created.



You can select default connecting as two way or one way, passing or nonpassing, straight or curved.

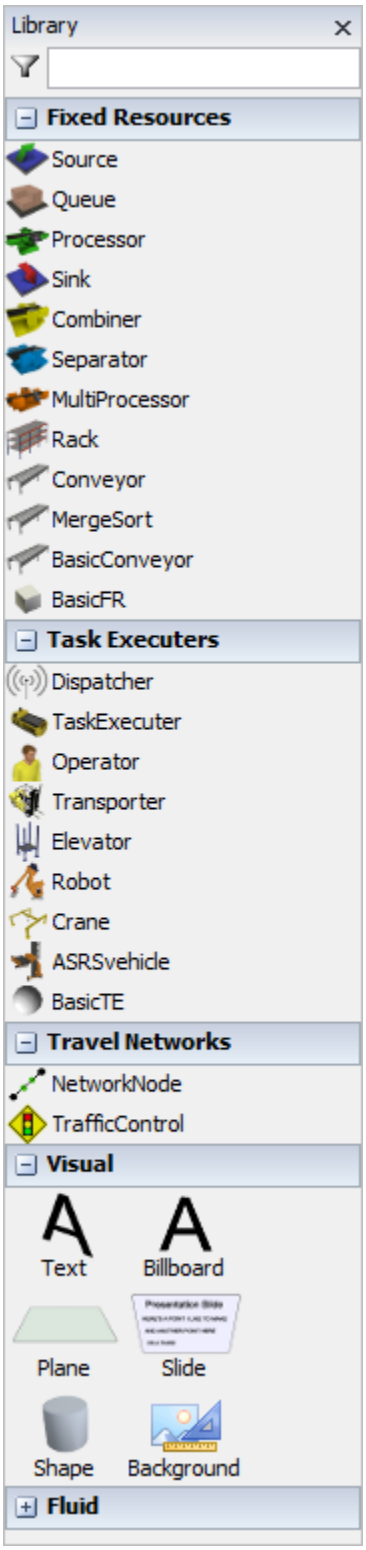
Node Draw Size

You can also change the size that network nodes are drawn by right clicking on the node and choosing **Node Draw Size** Network nodes are drawn a specific pixel size. They are unaffected by the zoom of the camera. That is why they appear to get larger as you zoom out and smaller as you zoom in relative to the other objects in your model.



Alternatively, you can left-click a NetworkNode and press the Ctrl+L or Ctrl+K key to increase and decrease NetworkNode sizes respectively.

Library Icon Grid



The library icon grid lets you drag objects into your model from one of FlexSim's standard libraries, or from custom-made libraries. If the grid is not already visible, you can access it through the **View menu > Drag-Drop Library**.

The objects are sorted into groups that can be expanded or collapsed by pressing the plus or minus buttons next to the group of objects. FlexSim's standard groups are Fixed Resources, Task Executers, Travel Networks, Visual, and Fluid. If you have loaded additional libraries, these libraries will be displayed at the top of the list in their own groups.

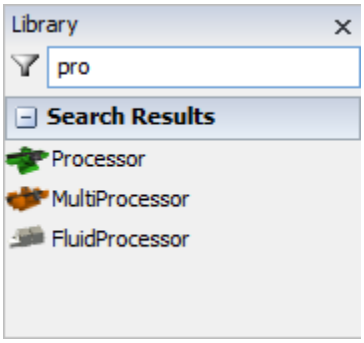
Creating Objects

There are two methods for creating objects from the Library:

- Click and hold on the object you would like to add to your model, drag it over a 3D view and release the mouse button at the location you would like to drop the object into the model.
- Click and release on an object in the Library to enter Create Mode. Then click anywhere in the 3D view to create new objects. To exit create mode, right click or press the Escape key.

Filtering

The Library Icon Grid can be filtered by name by entering text in the field.

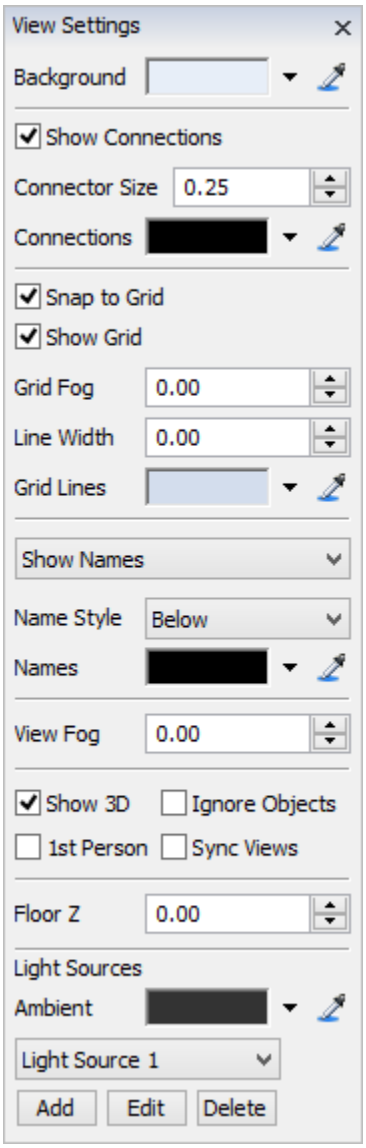


Filtering works for all Context Sensitive displays.

Context Sensitive

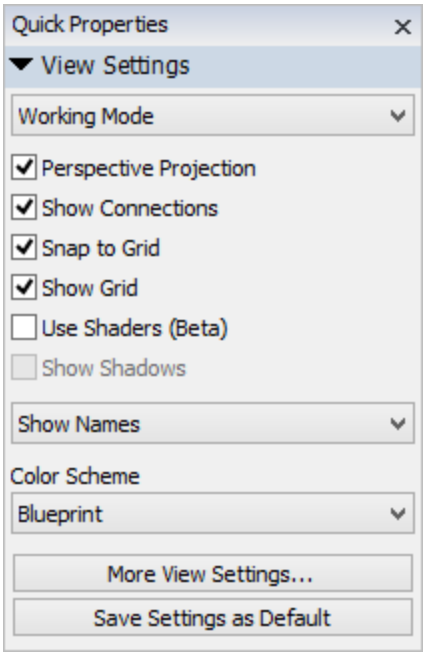
The Library Icon Grid is context sensitive. It will changed based on what the modeler is currently working on. It will display 3D shapes when in the FlowItem Bin and Animation Creator, code builder blocks while in the Code Editor and dashboard widgets while editing Dashboards.

View Settings



This dialog box can be opened by selecting **View** > **View Settings** from the main menu or clicking **More View Settings...** from the Quick Properties window. View Settings lets you configure the view's visual display settings. These settings apply only to currently active window and will be lost if that window is closed. If a window other than an 3D view is active, this window will be grayed out.

The Quick Properties window will display a few of the most common View Settings whenever a 3D view is active and no objects are selected in the view.



Mode - This option gives you two view setting presets. Switching to Presentation mode turns of Connections, the Grid, and Object Names.

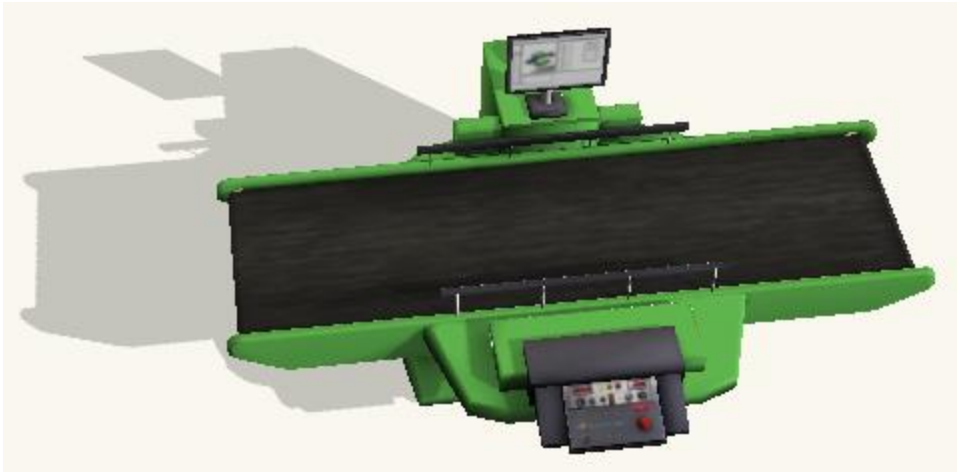
Perspective Projection - This option toggles the 3D view from being a Perspective View, to displaying as an Orthographic View. The Perspective View looks more real-world where as the Orthographic view has no depth.

Snap to Grid - If this box is checked, objects will automatically move to the nearest grid line when they are moved in the model. This is useful for placing objects in precise locations. Resizing of objects will also snap to the grid if this is checked.

Show Grid - If this box is checked, the grid will be drawn in the view window.

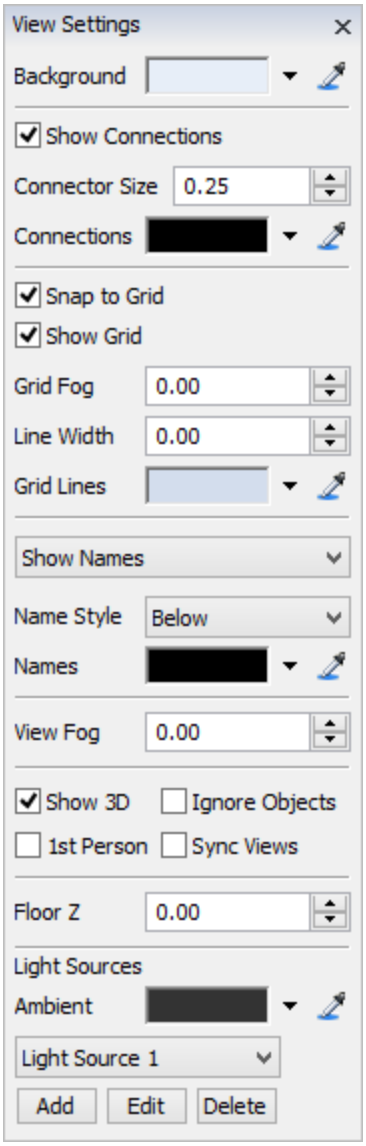
Use Shaders (Beta) - Uses shaders for drawing objects. This option is still in its Beta and is not guaranteed to function exactly the same in future releases. Using shaders does not work with Compatibility Mode.

Show Shadows - When shaders are enabled, you also have the option to have objects cast shadows on other objects. To cause objects to cast shadows on a "floor", create a plane Visual Tool and place it under your objects.



Color Scheme - These presets modify the 3D view colors.

Save Settings as Default - As described above, 3D View Settings are not saved and will be lost when the 3D view is closed. This will save your current View Settings as default for any new 3D views that are opened.



Background Color

Background Color - This option lets you select the color of the view window's background from a standard Windows color-choosing dialog box.

Connections

Show Connections - If this box is checked, the ports and port connection lines will be displayed in the view window. Hiding these connections often makes it easier to see what is happening in the model. If a model is slowing down, often un-checking this box will help speed it up.

Connector Size - This number sets how large the port connectors are on the object.

Connections Color - This value sets the color of the connector lines in the view.

Grid

Snap to Grid - If this box is checked, objects will automatically move to the nearest grid line when they are moved in the model. This is useful for placing objects in precise locations. Resizing of objects will also snap to the grid if this is checked.

Show Grid - If this box is checked, the grid will be drawn in the view window.

Grid Fog - This value lets you have the view's grid fade into the background color as it gets further from the viewpoint. Usually this is only useful on a perspective view. Set the value between 0 and 1, 0 meaning no fade, 1 meaning full fade.

Grid Line Width - This value sets the width of the view's grid lines.

Grid Line Color - This value sets the color of the grid lines.

Names

Show Names - This pull-down list allows you to choose whether names and stats are shown or hidden in the 3D view.

Name Style - This pull-down list allows you to choose where the name of the object will be drawn. Choose either below the object or above the object.

Names Color - This value sets the color of object names displayed in the 3D view.

Other Settings

View Fog - This value sets the view's fog value. View fog causes objects that are far away from the camera position to fade into the background color. Set the value between 0 and 1, 0 meaning no fog, and 1 meaning complete fog. This is usually only applicable to a perspective window.

Show 3D Shapes - If this box is checked, the 3D shapes (.3ds files or .wrl files) for all the objects in the model will be drawn in the view window. Some objects do not have 3D files associated with them, they are generally drawn directly with OpenGL. These objects will not be affected by this box.

1st Person - If this box is checked, the view window's mouse controls will be in first person mode. This means that the view will rotate around the user's view point, and not around a point in the middle of the screen. This mode is most useful when navigating in fly-through mode.

Ignore Objects - If this box is checked, the user will not be able to click on any objects in the view window. This is useful for navigating around a model that is completed, as the user will not be able to accidentally move any objects.

Sync Views - If this box is checked, all open view windows will be updated at the same time. If it is not checked, some windows may not be updated until an action is completed in a different window. Checking this box may cause the program to run a little slower.

Floor Z - This specifies the height (in model units) of the floor. This is useful when creating multi-floored models. Setting the Floor Z to the height of your model's 2nd floor will allow you to quickly and easily drag objects into the 3D view and have their Z position set to the Floor Z value.

Light Sources

These controls allow a user to add, edit and create light sources for the view window

Ambient Color - This value sets the color of light that is applied to all faces equally regardless of location.

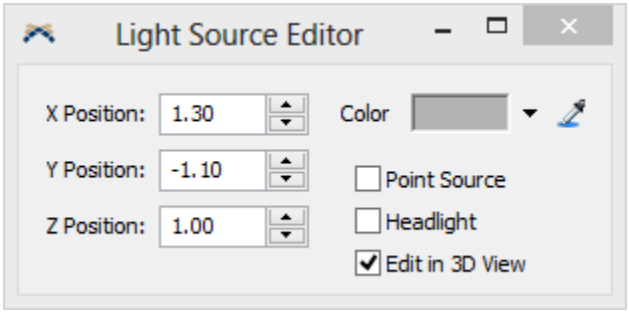
Light Sources - The pull-down list contains all of the light sources that are currently in the view window.

Edit - This button opens the Light Source Editor dialog box for the light currently showing the pull-down list.

Add - This button will create a new light source in the view window.

Delete - This button will delete from the view window the light source currently showing in the pull-down list. There must always be at least one light source in the model.

Light Source Editor



This dialog can be accessed by opening the View Settings and clicking the **Edit** button in the **Light Sources** section

X Position - This number is the position along the X axis of the model where this light source is located.

Y Position - This number is the position along the Y axis of the model where this light source is located.

Z Position - This number is the position along the Z axis of the model where this light source is located.

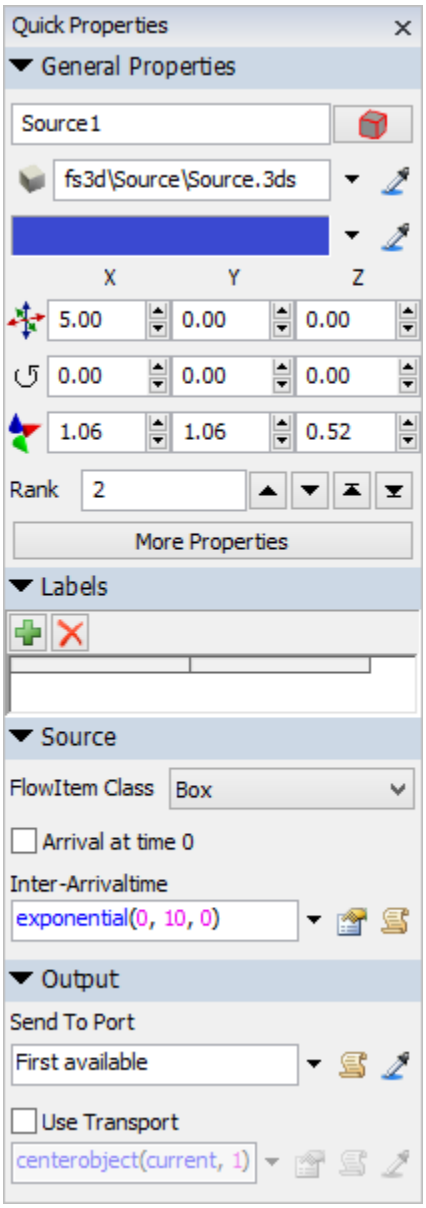
Color - This allows the user to choose the color of the light from this light source.

As point source - If this box is checked, the light generated by this light source appears to be coming from a specific point, relative to the camera. If it is not checked, the light appears to be coming from a given direction only.

Headlight - If this box is checked, the light light source will move with the camera acting as a headlight.

Edit in 3D View - If this box is checked, a lightbulb object will be displayed in the 3D view and allow you to move the light source to a desired position.

Quick Properties

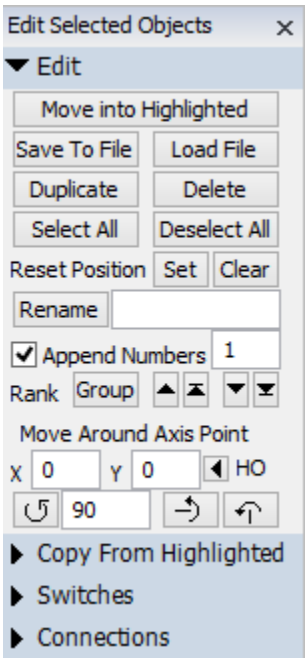


The Quick Properties window is a context sensitive window. Depending on what window you currently have active and what object is selected, the panels in the Quick Properties will change. We will not cover these context sensitive panels in this topic. For more information on these panels, see one of the following pages:

- Animation Creator
- Dashboard
- FlowItem Bin
- Orthographic/Perspective View
- Tree Window

If the Quick Properties have been closed, you can open them through the View menu.

Edit Selected Objects



This window can be accessed through the View menu > **Edit Selected Objects**.

It offers several options that are performed on the currently selected set of objects in that view window. To select a group of objects, hold the shift or control key down and drag a selection box around a group of objects. Objects in the selection set will have a red box drawn around them. The view's currently highlighted object (the last object you clicked on) will have a yellow box drawn around it.

Edit

Move into highlighted object - This option moves the selected objects (the ones with the red box) into the highlighted object (the one with the yellow box). This allows the highlighted object to be used as a container.

Save to file - The selected objects will be saved to a file with a .t extension that can later be reopened in FlexSim. This allows users to save and import parts of models as needed.

Load from file - This object loads a .t file into the currently highlighted object. The highlighted object then becomes a container for the imported objects.

Duplicate - This option creates an identical copy in the model of the selection set. All port connections are kept intact.

Delete - This option deletes the selected objects. This can also be done by highlighting one of the selected objects in a view and pressing the Delete or Backspace key.

Select All - This option adds all objects in the model to the selection set.

Deselect All - This option takes all objects in the model out of the selection set.

Set Reset Position - This option sets the reset position of each object in the selection set to its current position. When the model is reset, all objects with recorded reset positions will be moved, rotated, and sized to that position.

Clear Reset Position - This option removes the reset position of each object in the selection set.

Rename - Enter a name and press the rename button to rename all the selected objects.

Append Numbers - If this option is checked, the **Rename** button will also append numbers to the new name starting at the specified number and incrementing each renamed object by 1.

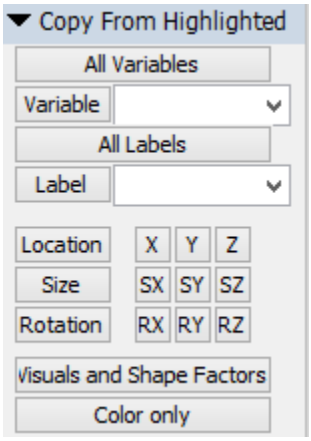
Rank - This series of buttons lets you group the objects together in the tree, as well as move them up and down.

Move Around Axis Point - This set of controls lets you manipulate the locations and rotations of the selected object set. All operations are done around an axis location in the model. You can either enter this in, or highlight an object in the model and click the "< HO" button to get that object's location. Enter an angle of rotation in degrees and press the button to rotate the selected set around the axis point. Click the or buttons to flip the selected objects over the axis point either vertically or horizontally, respectively.

Copy from Highlighted

These options copy information from the highlighted object (the one with the yellow box) to all of the objects in the selection set (the ones with red boxes).

If any of the selected objects have triggers that are toggled as C++ functions, the model will need to be recompiled before running it again.

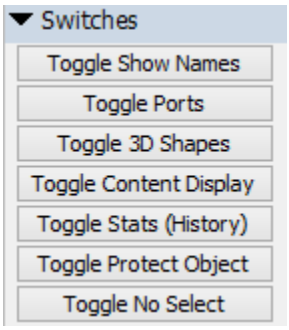


The following data can be copied:

- **All Variables -** Copies all variables (including all triggers) from the highlighted object to the selected objects.
- **All Labels -** Copies all labels from the highlighted object to the selected objects.
- **Variable or Label -** Single variable or label. Type in the name of the variable or label to be copied and press the "Variable" or "Label" button. Rather than typing the name explicitly, you can select an option from the dropdown list that shows you all the variables or labels of the highlighted object. The label dropdown list will be empty if the highlighted object has no labels on it.
- **Location, Size, Rotation -** These buttons allow you to copy spatial values from the highlighted object. You can copy the whole x/y/z size/location/rotation with one operation, or copy individual x, y, or z attributes.
- **Visuals and Shape factors -** This button copies visual information such as 3D shapes, 2D shapes and color. This also copies shape factors from the highlighted object.
- **Color only -** This button copies only the highlighted object's color to the selected objects.

Switches

These options allow you to turn on and off various settings on the selected objects. For all of the objects in the selection set, the chosen setting will be reversed (toggled). That is, if it is on it will be turned off, and if it is off it will be turned on.

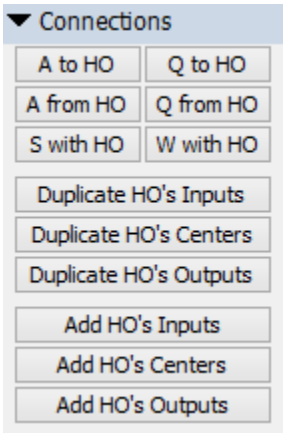


The available settings to toggle are:

- **Show Names -** Display the objects' names and stats.
- **Ports -** Display the objects' ports and port connections.
- **3D Shapes -** Display the objects' 3D shapes. Note that the Custom Draw Code trigger will not fire for objects with 3D shapes hidden in this way.
- **Content Display -** Display the content of the objects. If this is off, objects inside the main object will not be seen.
- **Stats (History) -** Activate stats collection for the objects.
- **Protect Object -** This locks the objects' location, size and rotation.
- **No Select -** Makes the object unclickable from the 3D view (object is still accessible through the Tree Window.

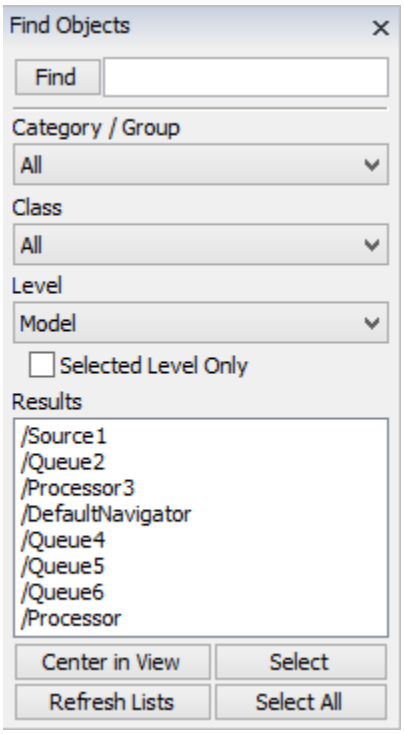
Connections

This panel allows you to make drag connections between the highlighted object and the selection set.



- **A to HO -** this option makes an 'A' drag connection from all objects in the selection set to the highlighted object.
- **Q to HO -** this option makes a 'Q' drag connection from all objects in the selection set to the highlighted object.
- **A from HO -** this option makes an 'A' drag connection from the highlighted object to all objects in the selection set.
- **Q from HO -** this option makes a 'Q drag connection from the highlighted object to all objects in the selection set.
- **S with HO -** this option makes an 'S' drag connection from all objects in the selection set to the highlighted object.
- **W with HO -** this option makes a 'W' drag connection from the highlighted object to all objects in the selection set.
- **Duplicate HO's Inputs -** this option destroys all the selected objects' input ports and creates copies of the highlighted object's input port connections to all objects in the selection set.
- **Duplicate HO's Centers -** this option destroys all the selected objects' center ports and creates copies of the highlighted object's center port connections to all objects in the selection set.
- **Duplicate HO's Outputs -** this option destroys all the selected objects' output ports and creates copies of the highlighted object's output port connections to all objects in the selection set.
- **Add HO's Inputs -** this option adds copies of the highlighted object's input port connections to all objects in the selection set.
- **Add HO's Centers -** this option adds copies of the highlighted object's center port connections to all objects in the selection set.
- **Add HO's Outputs -** this option adds copies of the highlighted object's output port connections to all objects in the selection set.

Find Objects



This window can be accessed through the View menu > **Find Objects**.

This utility lets you quickly find objects in your model based on the name or the type of object.

Find By Name

Type all or part of a name into the edit at the top and press Find to find an object in the 3D View or Tree view. This field is case-sensitive. "Queue" is different than "queue". Find will search through all of the objects in your model to find any object name that contains the specified text. ie searching for "Processor" would find both "Processor3" and "FluidProcessor4".

Find By Category/Group/Class

You can also select the library category or group (see Groups), and/or the class type, and/or a specific layer in the model that you want to find the objects in. When you select an option from the drop-down, the list at the bottom will be re-populated. Select an object in the list and press Center in View to go to that object in the 3D/Tree view.

Selected Level Only - Check this box to display only objects in the selected level. For example, if you have a Visual Tool in the model that contains three processors and you select the Level as Model, those three Processors will not be displayed in the Results list.

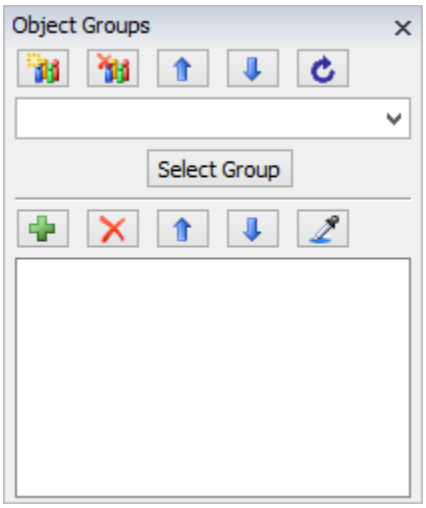
Center in View - Centers the object selected in the Results list in the 3D or Tree View.

Refresh Lists - Refreshes all lists including the Results List (useful when you've added an object to your model).

Select - Selects (red box) the selected object from the Results list.

Select All - Selects (red box) all of the objects in the Results list.

Groups



The groups utility lets you create and edit different groups in your model.

Group information is stored both in the /Tools/Groups folder and on each of the individual objects. A node is added to the object attributes called Groups with a coupling to the associated group. Objects may be included in multiple groups.

- Creates a new empty group.

- Removes the currently selected group.

- reorder the group.

- Reloads the list of groups.

Group Dropdown Menu - All groups are listed in this drop down menu. The name of the currently selected group is editable.

Select Group - Selects all the objects in the group (red box).

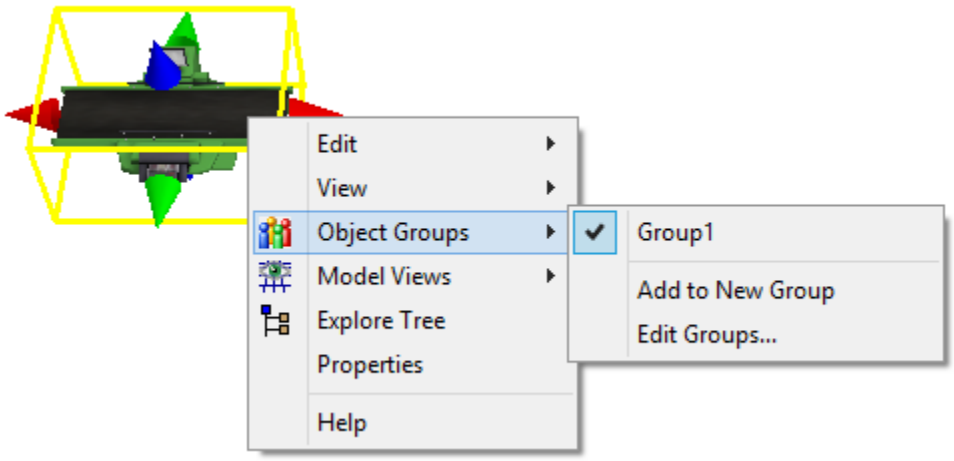
- Opens a menu allowing you to add currently selected items to the group or set the group to match the current model selection.

- Removes the selected(s) object from the group.

- reorder the members in the group.

- Click to enter "Sample" mode then select an object in the 3D View to add it to the group.

You may also add objects to groups through the right-click menu of the 3D View.

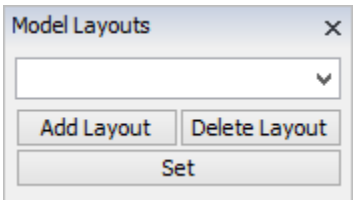


The current list of groups will appear at the top of the menu. Each group name will be checkable, so you can easily add objects to any number of groups

Add to New Group - Creates a new group and add's the object to the group. If multiple objects are selected (red box) this will add those objects to the group as well.

Edit Groups... - Opens the Groups window.

Model Layouts



This window can be accessed through the View menu > **Model Layouts**.

This utility lets you create and edit certain layouts for your model. Objects may be moved around in the view in order to create different layouts for different scenarios. For example, you may want to save a layout that represents a left-to-right flow chart of the steps in your model, where model flow proceeds always from left to right. Then you may want to save another layout that represents an actual factory layout. This window lets you do that. It is also useful for testing different potential layouts of your facility.

Layout information is stored on each of individual objects. A node is added to the object attributes called Layouts. For each created layout, a copy of the object's current spatial information (position, size, rotation) is stored.

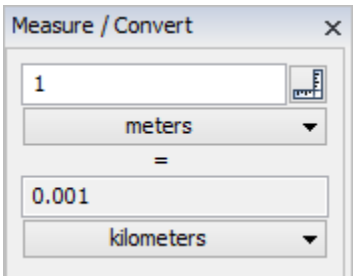
Layout List - Displays the list of current Model Layouts. Select a layout to immediately update your model that layout. You may rename a layout by entering a new name in this field.

Add Layout - Adds a new layout. When you add a layout, it will save the current layout of the model.

Delete Layout - Removes the selected layout.

Set - This will update the selected Model Layout to match the current layout of the model.

Measure / Convert



This window can be accessed through the View menu > **Measure Convert**.

The Measure/Convert tool allows you to take measurements of distances in your model as well as to convert length/time units. By default, the Measured units will be the model units as defined in the Model Settings window will be selected.

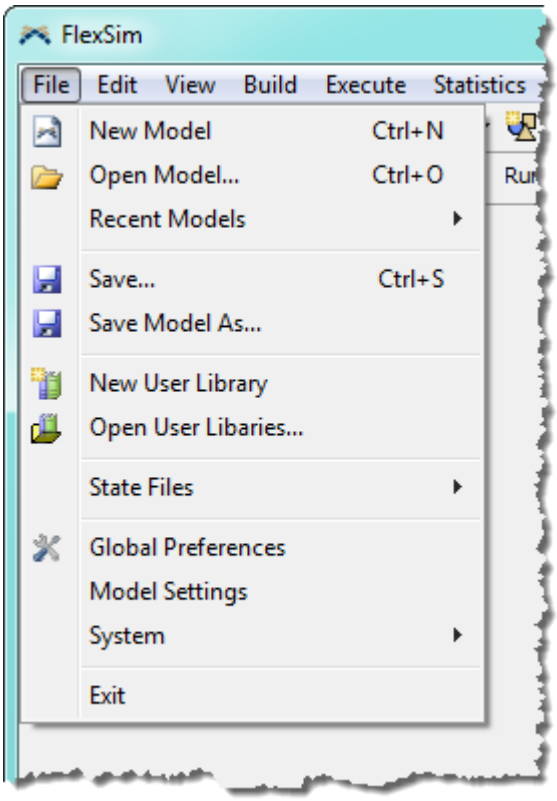
- Click this button to enter "Measure" mode. You can then click anywhere in the 3D view to define your start point, and click again to define the end point. The resulting distance will be displayed to the left.

In the first drop down (shown above as meters), you can select the units to "Measure" with or select a length, time or length per time unit to convert. You may type in your own value to the top field to make a conversion. The bottom field will be the result. The bottom drop down can be changed to convert to different units.

Main Menu and Toolbar

1. File Menu
2. Edit Menu
3. View Menu
4. Build Menu
5. Execute Menu
6. Statistics Menu
7. Tools Menu
8. Debug Menu
9. Help Menu
10. FlexSim Toolbar
11. Simulation Run Panel

File Menu



New Model - This option clears the current model so that a new one can be created. (A warning will be displayed asking if you would like to save your work.) The result is a blank FlexSim model, except that all opened user libraries will remain intact.

Open Model... - This option allows the user to choose a FlexSim Model File (.fsm extension) to edit. Any changes that have been made to the current model that have not been saved will be lost.

Recent Models - Shows a list of the latest FlexSim models to have been opened on this computer, with the most recently opened model at the top of the list.

Save... - This option saves the current model file (.fsm extension). Any changes that have been made to the current model will be saved.

Save Model As... - This option allows the user to save the model to a file. The most common file that is created is a FlexSim Model file (.fsm); the entire contents of the model tree will be saved. The two other file saving options are for a FlexSim XML file (.fsx – for more information see FlexSim XML) and a Tree file (.t – effectively identical to a .fsm file).

New User Library - This option creates a new user library and adds it to the library pane. For more information on custom libraries, see Creating Custom Libraries.

Open User Libraries... - This option loads one or more saved libraries, adding them to the library pane. If the library contains components for automatic install, a message will appear asking you if you want to install these components. Press OK to install these components. For more information on custom libraries, see Creating Custom Libraries.

State Files - This option allows the user to save the state (current model run) of the model, or load a saved state model to continue the run. Saving a state is helpful when your model is in the middle of a simulation run and you want to save it in its current state (all flowitems remain where they are and resources remain in their current state of operation), and then later load the FlexSim State file (.fst) and be able to continue running the simulation from that point.

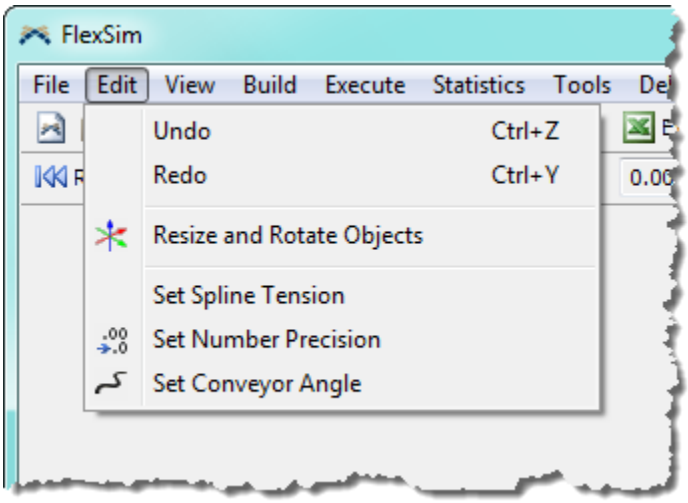
Global Preferences - Opens the Global Preferences window, which contains settings and options that affect the entire FlexSim application and are independent of any specific model file. For more information, see Global Preferences.

Model Settings - Opens the Model Settings window, with settings unique to the current FlexSim model. For more information, see Model Settings.

System - This option allows you to manually reload media, as well as disconnect any DLLs that have been linked to DLL toggled nodes.

Exit - This option will close FlexSim without saving any files. If the user wishes to save any changes, the appropriate file(s) should be saved before this option is selected.

Edit Menu



Undo - Erases the last change made to the model, reverting it to the state immediately prior to making the change.

Redo - Reverses the Undo effect.

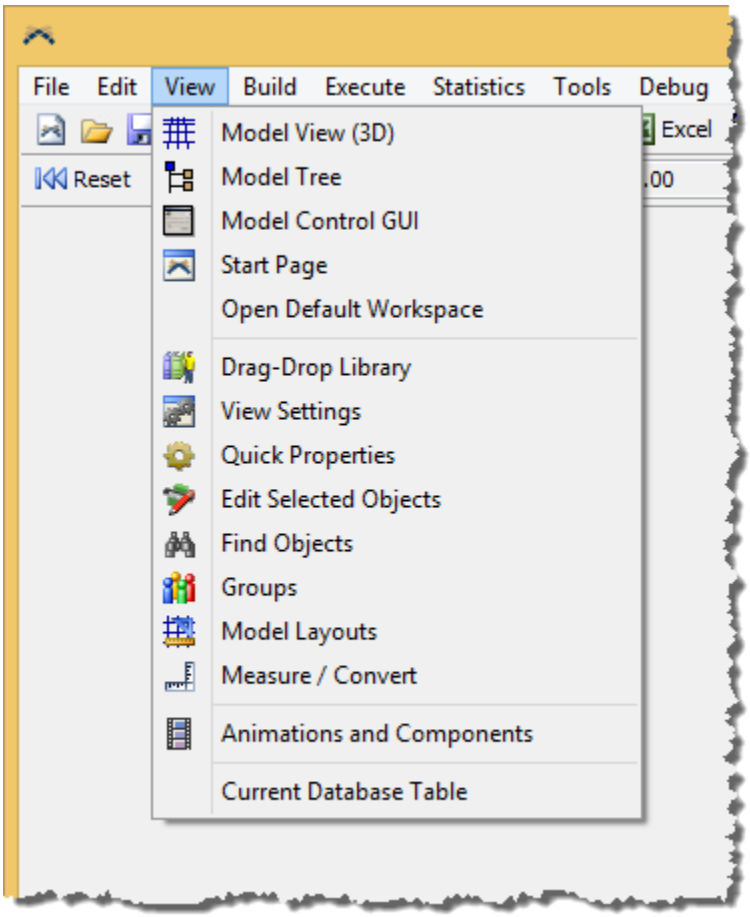
Resize and Rotate Objects - Clicking this option toggles whether or not object axes are shown in the Model View, which allows you to size and rotate model entities. By default, this option is enabled.

Set Spline Tension - This option brings up a dialog box to set the tension of splines in the model. This value should be between 0 and 1. A value of 0 will cause all splines to be totally straight passing directly through the nodes, and a value of 1 maximizes the curvature of the splines.

Set Number Precision - This option brings up a dialog box to set the number precision. This is the number of decimal points to be shown for numbers in the current FlexSim session.

Set Conveyor Angle - This option brings up a dialog box to set the conveyor angle. The conveyor angle determines how detailed curved conveyor sections are drawn. (A lower angle signifies a smoother curve.) This value only affects graphical performance, not model results.

View Menu



Model View (3D) - opens a new perspective view of the model.

Model Tree - opens a tree view that shows the model folder. This shows all of the objects that are in the model. The tree can be manipulated from this view.

Model Control GUI - opens the model's control GUI. This is a window that you can create yourself to do operations that are specific to your model. See Graphical User Interfaces for more information.

Start Page - opens FlexSim's start page, which is seen when you first open the FlexSim software. The start page provides options for creating a new model or opening an existing model, accessing the Getting Started guide and this users manual, adjusting global preferences, and licensing. It also shows the most recently edited FlexSim models, information on the currently installed license and FlexSim version, and some Internet-enabled active content.

Open Default Workspace - closes all of the active windows in FlexSim and then opens any windows saved as the default workspace from the Environment tab of the Global Preferences window.

Modeling Utilities

The following options will open various modeling utilities in one of FlexSim's panes. By default, the Drag-Drop Library and Quick Properties utilities are already open.

Drag-Drop Library - opens the Library utility. In most cases, this Library acts as an object library, with an icon grid of the available objects that can be dragged into the model.

View Settings - opens the View Settings utility.

Quick Properties - opens the Quick Properties utility. This utility provides quick access to object and flowitem properties that previously were only available in a properties window.

Edit Selected Objects - opens the Edit Selected Objects utility.

Find Objects - opens the Find Objects utility.

Groups - opens the Object Groups utility.

Model Layouts - opens the Model Layouts utility.

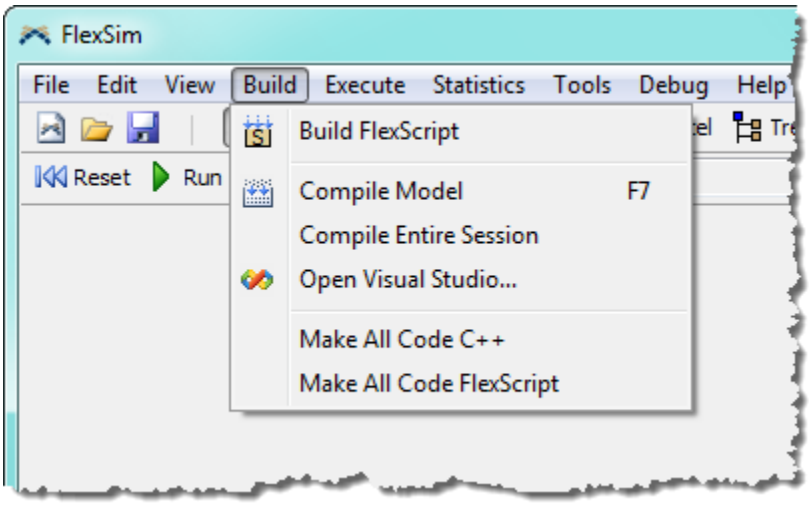
Measure/Convert - opens the Measure/Convert utility.

Animations and Components - opens the Animations and Components utility. This is the bottom window of the Animation Creator. Right click on an object and select **Edit > Animations** to open the Animation Creator.

Database

Current Database Table - This option brings up a window that shows the currently active database table that was opened or queried with the dbopen(), dbchangetable(), dbsqlquery() commands. Refer to the command summary for more information on these commands.

Build Menu



Build FlexScript - Builds all FlexScript code.

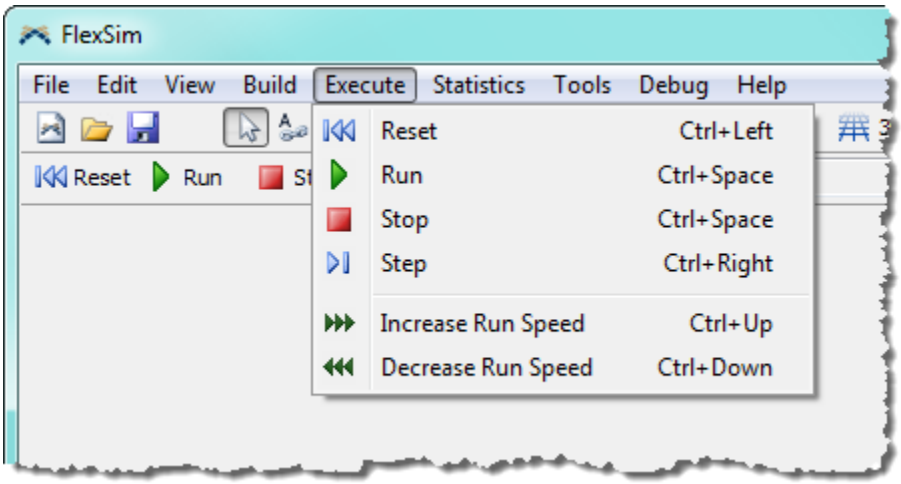
Compile Model - Compiles all C++ code in the model. This feature can also be accessed any time by pressing F7.

Compile Entire Session - Compiles all C++ code in the main tree. You usually will not need to do this unless you are developing your own application with FlexSim.

Open Visual Studio... - Opens Microsoft Visual Studio.

Make all Code C++/FlexScript - There are two options to make all code either C++ or FlexScript. We provide these options so that modelers can have both the ease of use of FlexScript (code works immediately when editing in FlexScript, without having to compile) as well as the run-speed of C++ (since it is compiled, it runs much faster than FlexScript). While in the model building phase you can use FlexScript, so that your code is interpreted immediately after you write it. Then, once your model is ready to run, you can choose the **Build > Make All Code C++** option, compile, and run to get the speed of C++.

Execute Menu



Reset - This option resets the current model file and prepares it to run. Same as selecting the Reset button on the simulation run panel. (Also accessible by holding down *Ctrl* and pressing the *left arrow*.)

Run - This option runs the current model file. Same as selecting the Run button on the simulation run panel. (Also accessible by holding down *Ctrl* and pressing the *Spacebar* to toggle between run and pause.)

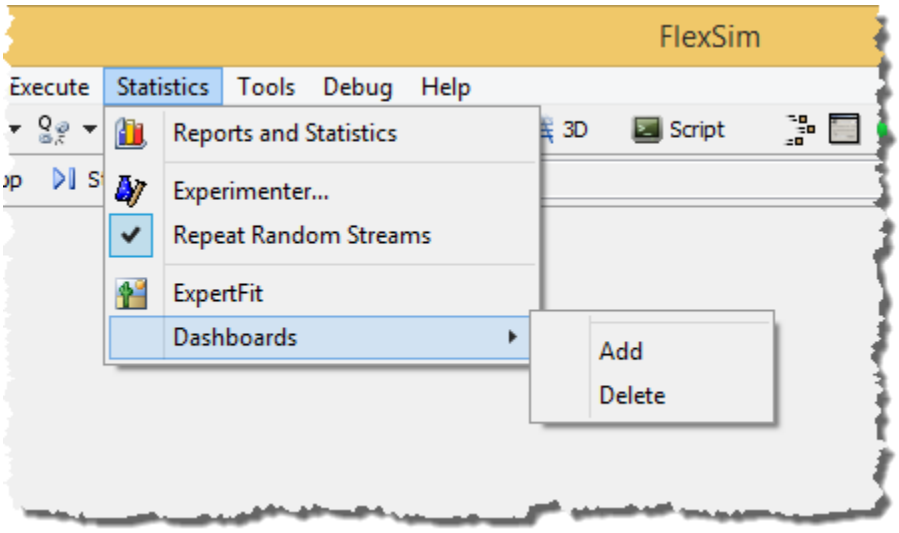
Stop - This option stops the current model. Same as selecting the Stop button on the simulation run panel.

Step - This option pushes the model forward to the next event on the event list. Same as selecting the Step button on the simulation run panel. (Also available by holding down *Ctrl* and pressing the *right arrow*.)

Increase Run Speed - This option increases the run speed of the model by increasingly greater degrees. Same as moving the slider bar to the right on the simulation run panel. (Also available by holding down *Ctrl* and pressing the *up arrow*.)

Decrease Run Speed - This option decreases the run speed of the model by increasingly greater degrees. Same as moving the slider bar to the left on the simulation run panel. (Also available by holding down *Ctrl* and pressing the *down arrow*.)

Statistics Menu



Reports and Statistics - This option opens the Reports and Statistics window. It allows the modeler to create various reports based on statistics gathered during a model run. These reports can include information about flowitem throughput, staytime, state history and other data that the modeler can select or customize.

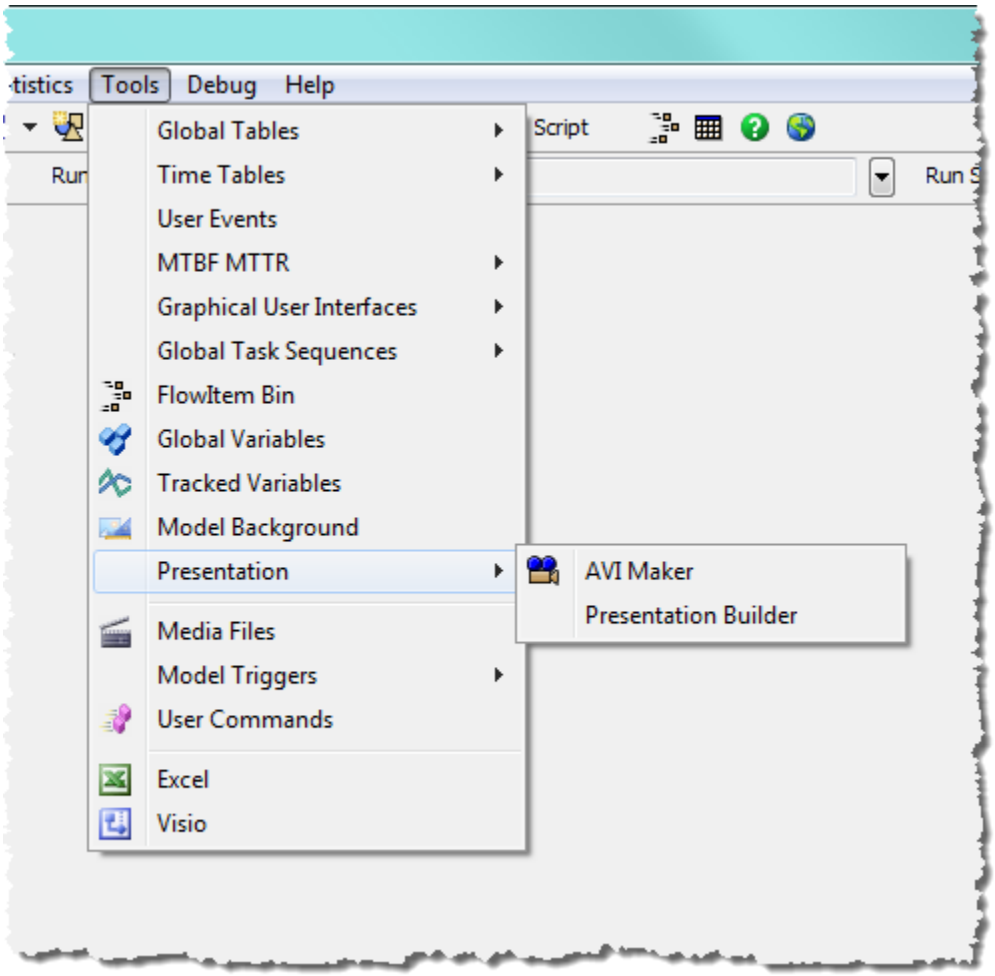
Experimenter - This option brings up the Simulation Experiment Control input window. The Experimenter is used to perform multi-run multi-scenario analyses of your model. This is also how you access FlexSim's Optimizer, which is used to find the optimum values for a set of user-defined decision variables so as to either minimize or maximize a user-defined objective function within the constraints defined by the user. The Optimizer, using OptTek's **OptQuest** engine, requires a special license sold as an add-on to the basic FlexSim software license.

Repeat Random Streams - This option repeats the random streams every time you reset and run the model. To make each run different, unselect this option.

ExpertFit - This option will start the ExpertFit software, developed by Averill Law & Associates. A complete user manual is available within the ExpertFit software. ExpertFit is a program used to determine the best statistical probability distribution to match your input data.

Dashboards - Contains a list of your current Dashboards and allows you to add new Dashboards. For more information, see Dashboards.

Tools Menu



Global Tables - a sub-menu which provides functionality for adding, deleting and editing Global Tables in your model.

Time Tables - a sub-menu which provides functionality for adding, deleting and editing Time Tables in your model.

User Events - opens the User Events window.

MTBF MTTR - a sub-menu which provides functionality for adding, deleting and editing MTBF MTTR objects in your model.

Graphical User Interfaces - a sub-menu which provides functionality for adding, deleting and editing Graphical User Interfaces in your model.

Global Task Sequences - a sub-menu which provides functionality for adding, deleting and editing Global Task Sequences in your model.

Flowitem Bin - opens the FlowItem Bin in a center pane, allowing the user to edit the master Flowitem list for the model.

Global Variables - opens the Global Variables window, allowing you to add variables to the model that can be accessed by any object in the model.

Tracked Variables - opens the Tracked Variables window. Tracked variables are used for recording output data about the model. They can be graphed as a line chart (if time-weighted) or a dot plot in the Dashboard. By default, FlexSim automatically calculates work-in-progress and time-in-system using tracked variables.

Model Background - opens the Background Drawing Wizard, which allows you to add a background picture (*.bmp, *.jpg) or AutoCAD (*.dxf, *.dwg) drawing.

Presentation

AVI Maker - opens the AVI Maker.

Presentation Builder - opens the Presentation Builder in a center pane, which allows you to develop fly-through presentations.

Media Files - opens the Media File window, which allows you to add 3D shapes and images to be pre-loaded into the model.

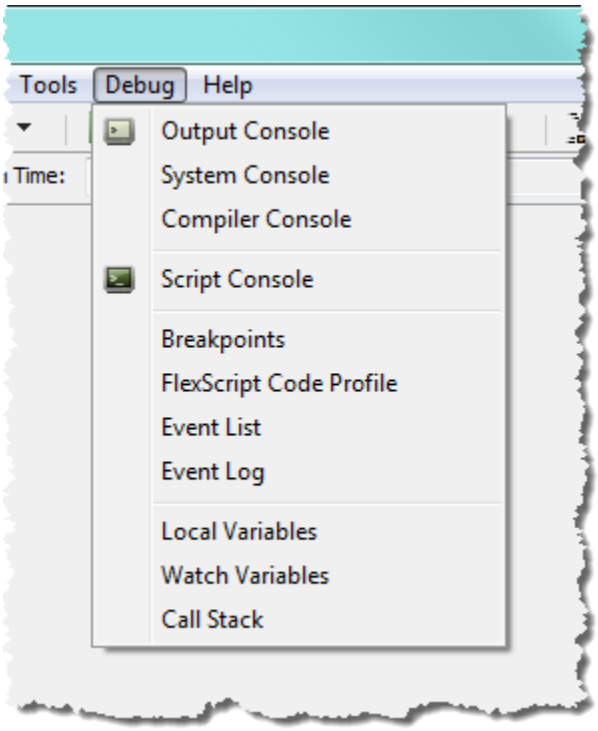
Model Triggers - a sub-menu which provides options to open the relevant Model Trigger window.

User Commands - opens the User Commands window, which allows the user to add, delete, and edit custom commands in the model.

Excel - opens the Excel interface, which allows the user to define the parameters for Excel imports and exports. This option is the same as pressing the Excel button on the toolbar.

Visio - opens the Visio Import window.

Debug Menu



Output Console - This option opens a pane where output information is shown. You can print your own information to the output console using the commands: pt(), pr(), pd(), pf(), etc. For more information on these commands, refer to the command summary.

System Console - This option opens a pane where information about the status of the program engine is printed. Unhandled exceptions and other errors will be printed to this console.

Compiler Console - This option opens a pane where information is printed while the model is compiled. This shows the status of each step of the compilation process. This console also shows any FlexScript errors when you Build FlexScript.

Script Console - This option opens the scripting console. This pane is an interactive window to execute FlexScript commands manually. See Scripting Console for more information.

Breakpoints - This option opens the Breakpoints pane. See Breakpoints for more information.

FlexScript Code Profile - This option opens the FlexScript Code Profile pane, which lists all the FlexScript functions defined in the model, ordered by how much time the model spent executing them. This is an excellent way to find out how the computer is spending its time and might lead to ideas on how to make the model faster. See Code Profiler for more information.

Event List - This option opens the Event List window, a sorted list of all pending events. See Event Lists for more information.

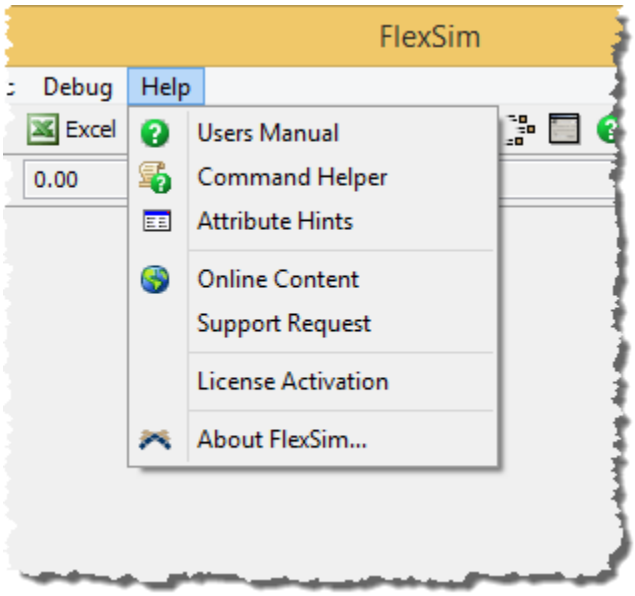
Event Log - This option opens the Event Log window, a sorted list of all events that have taken place. See Event Log for more information.

Local Variables - This option opens the Local Variables pane, which shows you the current values of any locally defined variables. See Local Variables for more information.

Watch Variables - This option opens the Watch Variables pane, which allows you to specify other variables or expressions that you want to see, such as global variables. See Watch Variables for more information.

Call Stack - This option opens the Call Stack pane, which shows the current call stack, a function call history. See Call Stack for more information.

Help Menu



Users Manual - opens this FlexSim user manual.

Command Helper - opens the Command Helper pane that allows you to reference any FlexSim command. Provides details, parameters, and examples of each command.

Attribute Hints - opens the Attribute Hints window. This list shows all of the FlexSim attributes and their meanings.

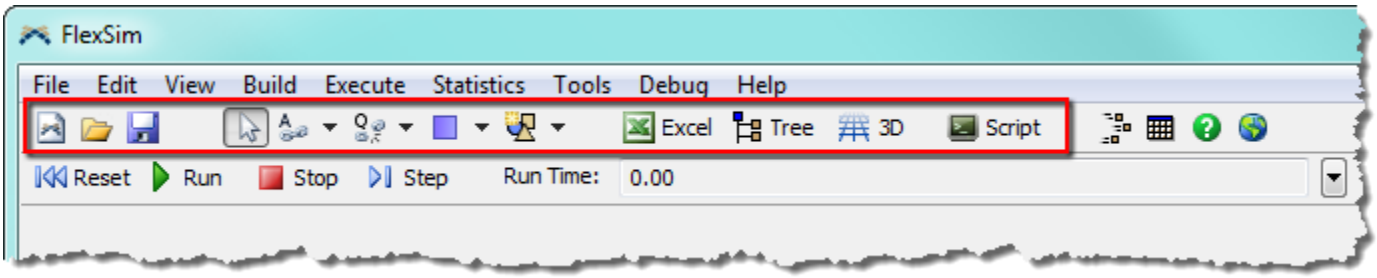
Online Content - uses an internal browser to view and download online content through FlexSim including modules, 3D shapes, images, and sample models.

Support Request - opens the FlexSim Support Request page in a web browser, allowing you to submit a support ticket.

License Activation - opens the License Activation window.

About FlexSim... - opens a splash screen giving you information about FlexSim. It displays the license status, the version of FlexSim currently running, graphics card info, and contact information.

FlexSim Toolbar



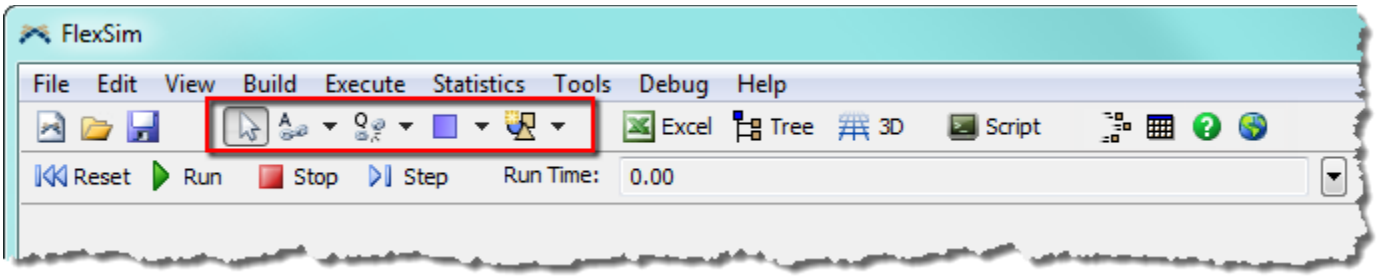
New: closes the current model and allows you to begin building a new one. A dialog box will appear asking if you'd like to save before creating a new model.

Open: allows you to open a previously saved model (.fsm file). A dialog box will appear asking if you'd like to save before creating a new model.

Save: allows you to save the current model. If the model has never been saved before, you will be asked to specify where the file will be saved.

Mode Toolbar

The mode toolbar lets you toggle between different modes in your 3D view. You enter a mode by pressing the appropriate mode button, and then can exit the mode either by pressing the Esc key or by pressing a different mode button. Alternatively, you can hold down keys as defined in the keyboard interaction page.



Standard Mode - This mode is the standard mode for your view. Here you can move objects around, size them, etc. You can return to this mode at any time by pressing the Esc key.

Connect Objects - To connect two objects in this mode, click-and-hold on one object, drag the mouse to the other object, and release the mouse button on that object. You can also create multiple connections in series in this mode. To do this, click an object, then click another object, then click another object, and so on. This works the same way as holding down the A key.

Connect Center Ports - This mode works the same way as the Connect Objects mode, except that it connects center ports instead of input/output ports. This works the same way as holding down the S key.

Extended Connect - This mode works the same way as the Connect Objects mode, except that it does an extended connect that varies depending on the objects clicked. This works the same way as holding down the D key. For more information on extended connections (also known as "context sensitive" connect), see the keyboard interaction section.

Disconnect Objects - To disconnect two objects in this mode, click-and-hold on one object, drag the mouse to the other object, and release the mouse button on that object. You can also delete multiple connections in series in this mode. To do this, click an object, then click another object, then click another object, and so on. This works the same way as holding down the Q key.

Disconnect Center Ports - This mode works the same way as the Disconnect Objects mode, except that it disconnects center ports instead of input/output ports. This works the same way as holding down the W key.

Extended Disconnect - This mode works the same way as the Disconnect Objects mode, except that it does an extended disconnect that varies depending on the objects clicked. This works the same way as holding down the E key. For more information on extended disconnections (also known as "context sensitive" disconnect), see the keyboard interaction section.

New Selection - You can create selection sets to have operations apply to a whole set of objects. To select objects in this mode, drag a box around the objects that you want selected. To select none, click the background of the view in this mode. This works the same way as holding down the Shift key.

Toggle Selection - You can create selection sets to have operations apply to a whole set of objects. To select objects in this mode, drag a box around the objects that you want selected. Any selected objects in the box will become unselected. Any unselected objects in the box will become selected. This works the same way as holding down the Ctrl key.

Create Objects - Once you are in this mode, you can simply click on an object in the library, and then each time you click the mouse's left button in your ortho view, a new instance of that object will be created. This works the same way as holding down the F key.

Create and Connect Objects - This mode is similar to the previous mode, except that when a new object is created, it will be connected with the previous object that was created ('A'). This works the same way as holding down the R key.

Excel**:** opens the Excel interface dialog box.

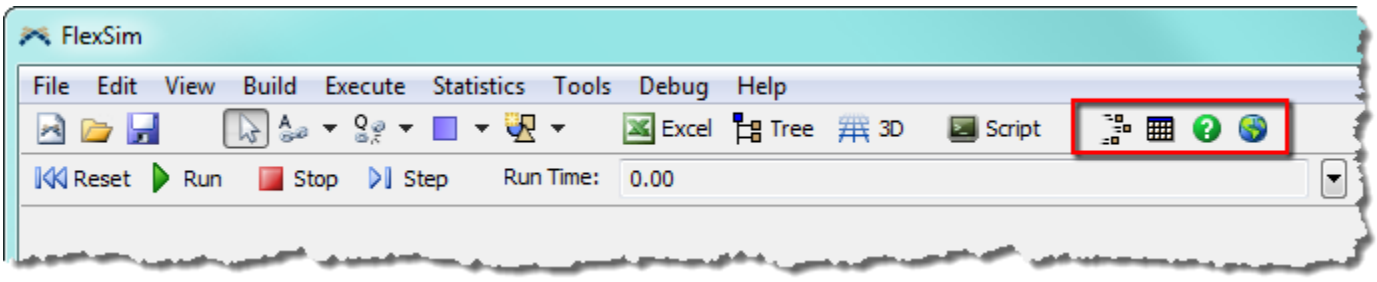
Tree**:** opens a tree view of the model. This is useful to see all of the objects that are currently in the model, even those that cannot be seen in the Model View. Objects' attributes can also be edited here, however, it is recommended that object properties be changed through the object's Properties dialog window or Quick Properties, and not through the tree.

3D**:** opens a new perspective view of the model.

Script**:** opens a script console. Here you can execute FlexScript commands to change or get information from your model.

Custom Toolbar

This section of the toolbar can be customized with any options from the main menu. To customize this section, go to **File > Global Preferences** and select the Customize Toolbar tab. The following are a few of the options that can be added to this toolbar:



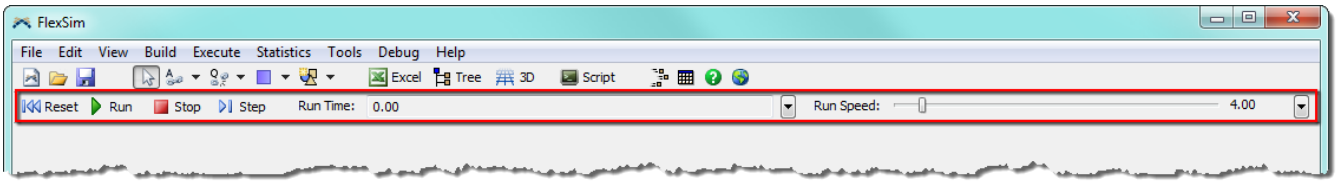
Flowitem Bin: opens the FlowItem Bin editor. In this window, the user can edit the types of flowitems currently available in the FlowItem Bin. Attributes of the flowitems, such as size, shape, and color, can be changed; packing methods can be specified; and animations can be added. New flowitem classes can also be created, and unused ones can be deleted.

Model Control GUI: opens the model's control GUI. This is a window that you can create yourself to do operations that are specific to your model. For more information, see Graphical User Interfaces.

Getting Started: opens the FlexSim Getting Started help section.

Sample Models: opens the Models section of the FlexSim Downloads web site in a web browser.

Simulation Run Panel


FlexSim
File Edit View Build Execute Statistics Tools Debug Help
Excel Tree 3D Script
Reset Run Stop Step Run Time: 0.00 Run Speed: 4.00

This panel is used to control a model's run. Most of the functions are available through the Execute Menu in the main menu. The Simulation Run panel is found at the top of the main window in FlexSim, and we describe it here in three sections:

Buttons

Reset: resets the model. The OnReset function is called for each object in the model. This should always be selected before running a model.

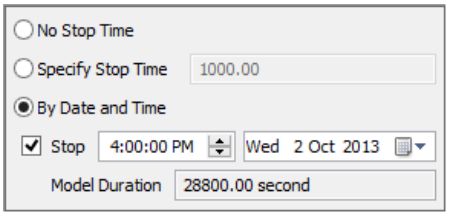
Run: starts the model running. The model clock will continuously advance until the model is stopped or the event list is empty.

Stop: stops the model while it is running. It also updates the states of all objects in the model. The model is not reset. It can be started again from the exact point in time that it was stopped.

Step: sets the model clock to the time of the next event that needs to occur. That event then happens. This enables users to step through the execution of a model one event at a time. When many events are scheduled to occur at the same time, there may be no visible changes in the model when this option is selected.

Run Time: Displays the model's current run time.

Time/Stop Time



No Stop Time: select this to have the model run indefinitely. The model can still be stopped if the "Stop" button is clicked or the event list is empty.

Specify Stop Time: select this to specify the number of time units the model will run for before stopping. The model may also be stopped at a given time using a picklist option on any relevant object in the model. In the "OnEntry" or "OnExit" trigger, the "Stop the Model Run" option can be used to dynamically stop the model run.

By Date and Time: If this option is selected, the Run Time field will display the model's current date and time. The model start date and time is defined in your Model Settings. specifies a model run duration based on a specific date and time. If the "Stop" box is not checked, the model will run indefinitely from the start time; if a stop time has been chosen, the model duration becomes the difference between the two times. The "By Date and Time" feature will work with Time Tables who are using the Graphical Time Table.

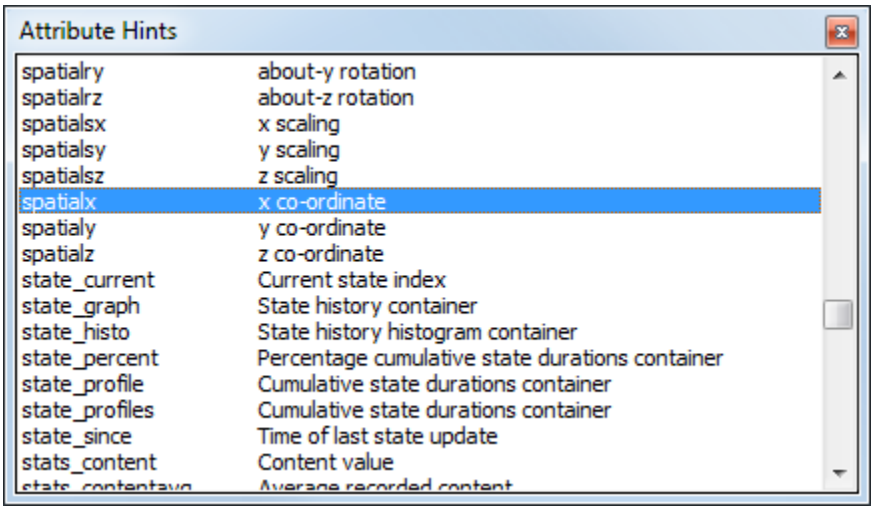
Speed

Speed slider: this slider defines the number of model time units that FlexSim will try to calculate per second of real time. The actual result may fall short of this value if the model requires too much processing at each event. This value can also be changed by clicking the black arrow to the right and entering a number manually.

General Windows

1. Attribute Hints
2. Model Settings
3. Global Preferences Window
4. Tree Browse Dialog
5. Database Table View
6. Table Editor
7. Find / Replace

Attribute Hints

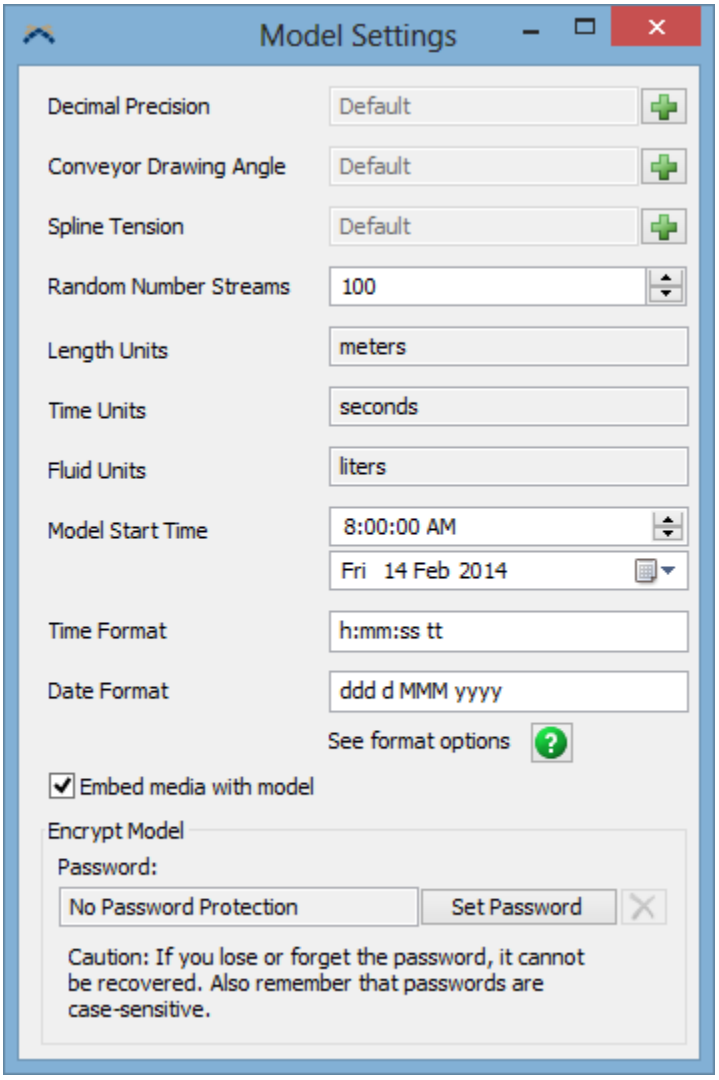


This table view is accessed from the Help menu. The attribute hints window shows the list of special FlexSim attributes and their meaning. Click inside the window and start typing the name of an attribute you would like to know about to have the window will automatically scroll to that attribute.

For more information about attributes, see the View Attributes Reference and GUI Reference pages.

Model Settings

This table view is accessed from the File menu.



Decimal Precision- This option controls the precision of editable values seen in the interface.

Conveyor Drawing Angle- This option controls how straight conveyor curves are. A value close to 0 produces smooth conveyors, while a value close to 90 produces square conveyors. Flowitems always follow the curved path.

Spline Tension- This option controls how straight the lines in a curved network node path are. A value close to 0 will produce straight lines. A value close to 1 will produce curved lines.

Random Number Streams- This option controls the range (between 0 and the specified value) of random number streams that will be initialized by Flexsim when the model is reset.

Length Units- This displays the length units of the model.

Time Units- This displays the time units of the model.

Fluid Units- This displays the fluid units of the model.

Model Start Time- This displays the global model start time and date that is used in various objects like the TimeTable.

Time Format- This specifies what format the Model Start Time should be displayed in. See format options.

Date Format- This specifies what format the Model Start Date should be displayed in. See format options.

Embed media with model- This option allows you to embed all of the 3D and image files associated with your model into the model file.

Encrypt Model- This area allows you to set a password for your model, so that only those with the password can open it. There is no way to recover lost passwords. To set a password, click the **Set Password button**. Then enter the desired password and click the **Enter Password** button. Enter the password again and click the **Confirm Password** button. To remove a password, click the ✕ button.

Date/Time Format Strings- The following is a list of characters or elements that may be used in a date/time format.
Body text can be added to the format string by enclosing the text in single quotes.

Note: Time formats and date formats are exclusive and may not be mixed.

Time Format

Element	Description
h	The one- or two-digit hour in 12-hour format.
hh	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
m	The one- or two-digit minute.
mm	The two-digit minute. Single-digit values are preceded by a zero.
s	The one- or two-digit second.
ss	The two-digit second. Single-digit values are preceded by a zero.
t	The one-letter AM/PM abbreviation (that is, AM is displayed as "A").
tt	The two-letter AM/PM abbreviation (that is, AM is displayed as "AM").

Date Format

Element	Description
d	The one- or two-digit day.
dd	The two-digit day. Single-digit day values are preceded by a zero.
ddd	The three-character weekday abbreviation.
dddd	The full weekday name.
M	The one- or two-digit month number
MM	The two-digit month number. Single-digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
yy	The last two digits of the year (that is, 1996 would be displayed as "96").
yyyy	The full year (that is, 1996 would be displayed as "1996").

Global Preferences Window

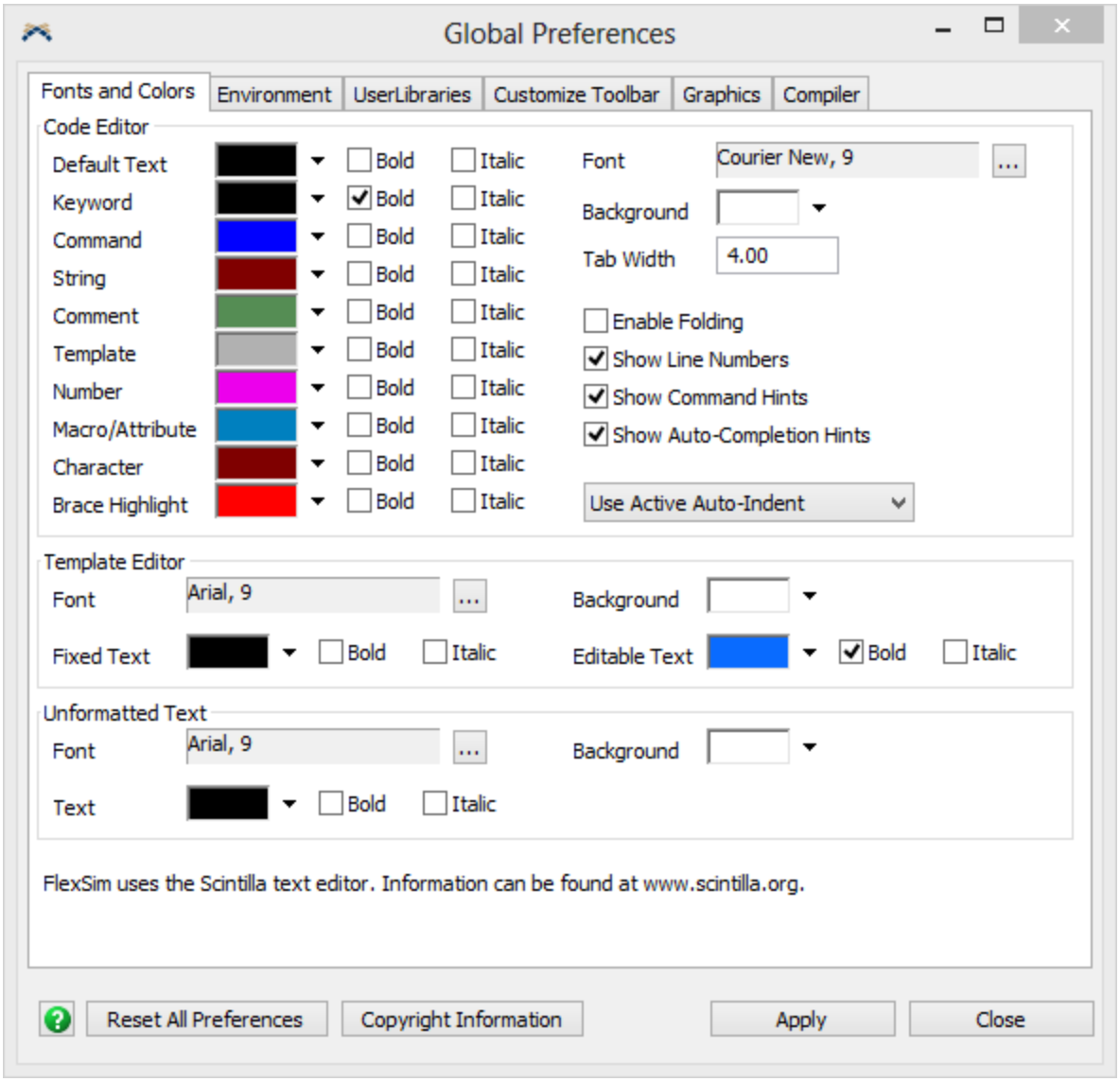
This table view is accessed from the File menu. The global preferences dialog window specifies default user preferences for FlexSim. These preferences are saved across several models and remembered when FlexSim is closed and reopened. The dialog window is split into six tabs.

Tabs

- **Fonts and Colors**
- **Environment**
- **User Libraries**
- **Customize Toolbar**
- **Graphics**
- **Compiler**

Fonts and Colors

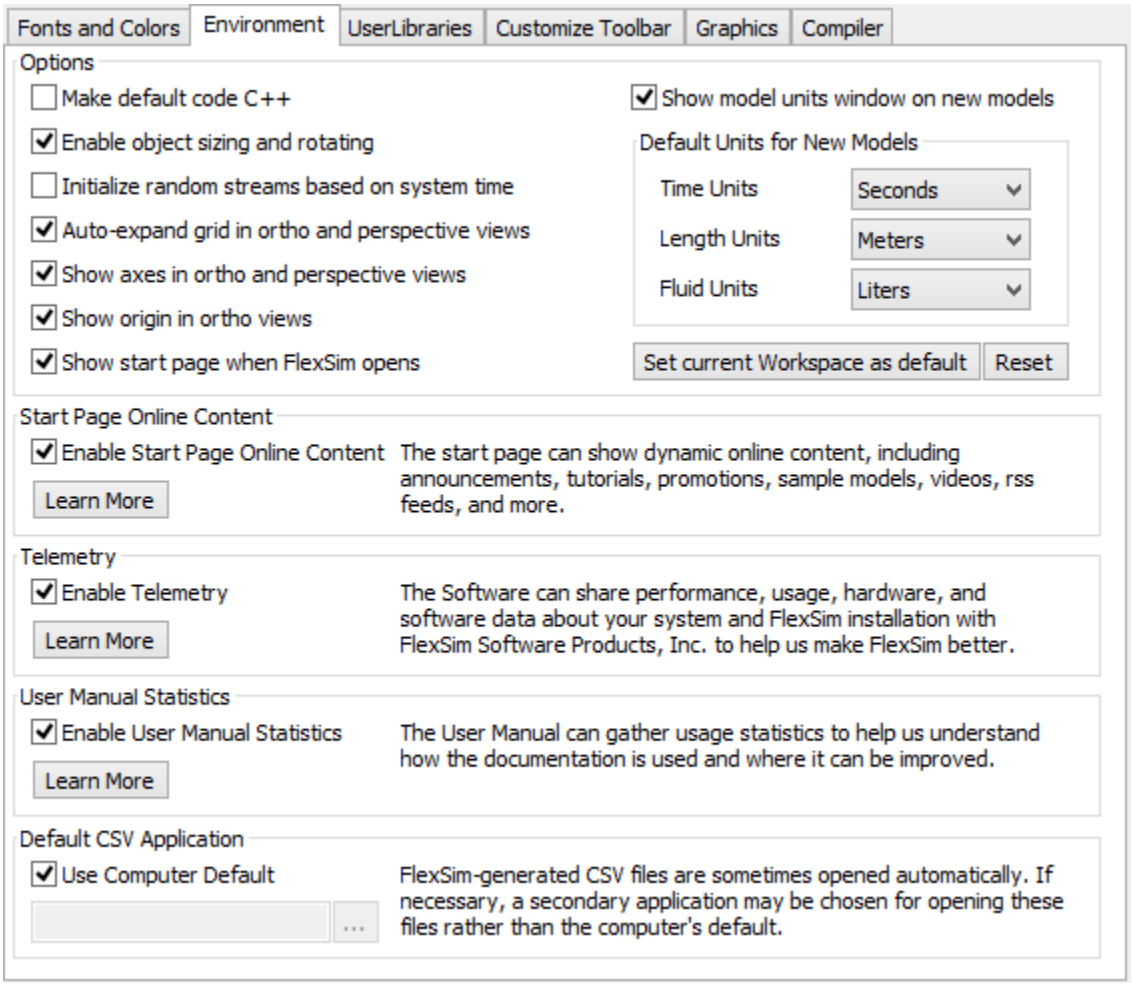
The Fonts and Colors tab specifies syntax highlighting and other settings used in FlexSim's implementation of the Scintilla code editor. You can also specify settings and colors for the template editor, as well as for unformatted text (which is used in multi-line unformatted text controls like the user description portion of the Properties window).



For more information on the Scintilla code editor, go to www.scintilla.org.

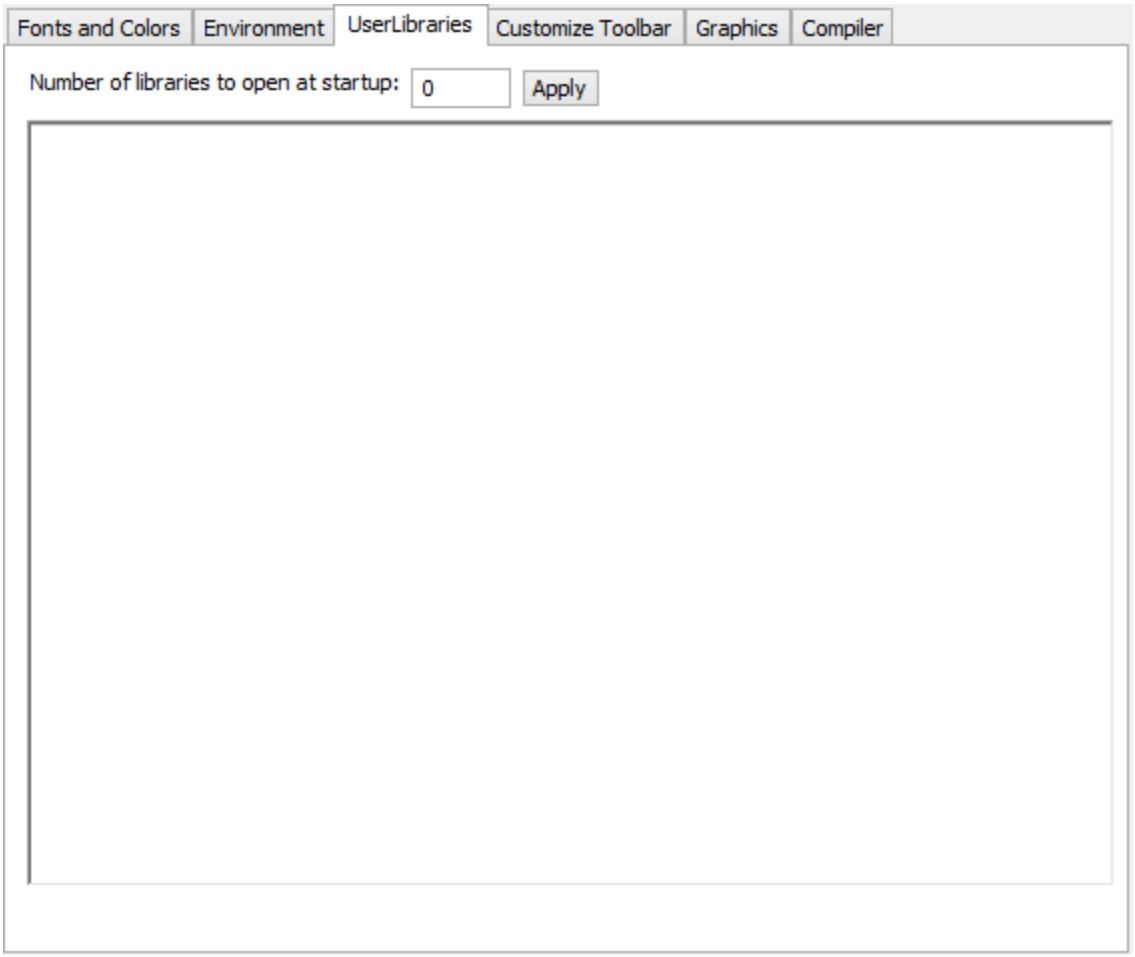
The Scintilla text editor is under copyright 1998-2003 by Neil Hodgson <neilh@scintilla.org>. All Rights Reserved.

Environment


Fonts and Colors
Environment
UserLibraries
Customize Toolbar
Graphics
Compiler
Options
Make default code C++
Enable object sizing and rotating
Initialize random streams based on system time
Auto-expand grid in ortho and perspective views
Show axes in ortho and perspective views
Show origin in ortho views
Show start page when FlexSim opens
Show model units window on new models
Default Units for New Models
Time Units
Seconds
Length Units
Meters
Fluid Units
Liters
Set current Workspace as default
Reset
Start Page Online Content
Enable Start Page Online Content
Learn More
The start page can show dynamic online content, including announcements, tutorials, promotions, sample models, videos, rss feeds, and more.
Telemetry
Enable Telemetry
Learn More
The Software can share performance, usage, hardware, and software data about your system and FlexSim installation with FlexSim Software Products, Inc. to help us make FlexSim better.
User Manual Statistics
Enable User Manual Statistics
Learn More
The User Manual can gather usage statistics to help us understand how the documentation is used and where it can be improved.
Default CSV Application
Use Computer Default
FlexSim-generated CSV files are sometimes opened automatically. If necessary, a secondary application may be chosen for opening these files rather than the computer's default.

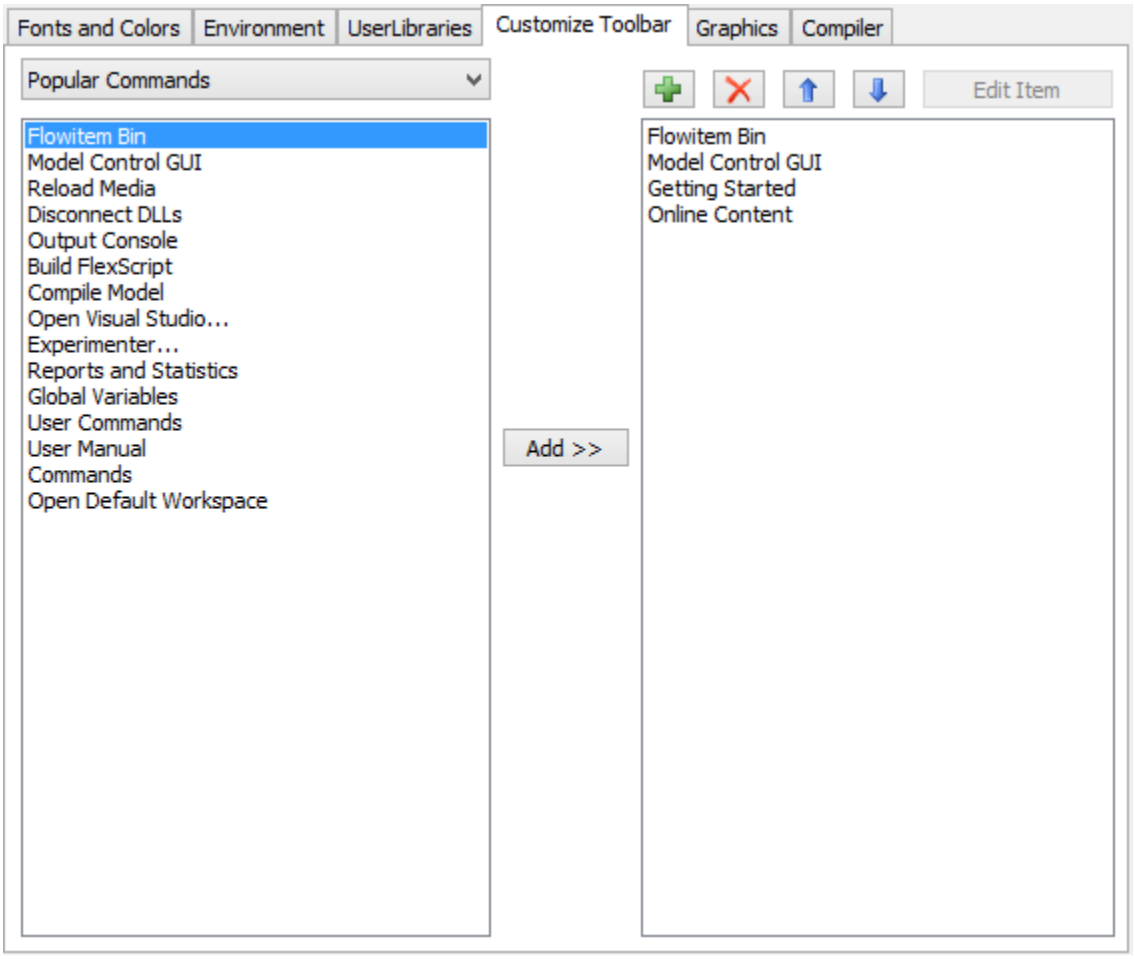
In this tab page you can specify various settings such as whether you want code to be C++ or FlexScript by default, various grid settings in the ortho and perspective view, excel DDE settings, etc.

User Libraries



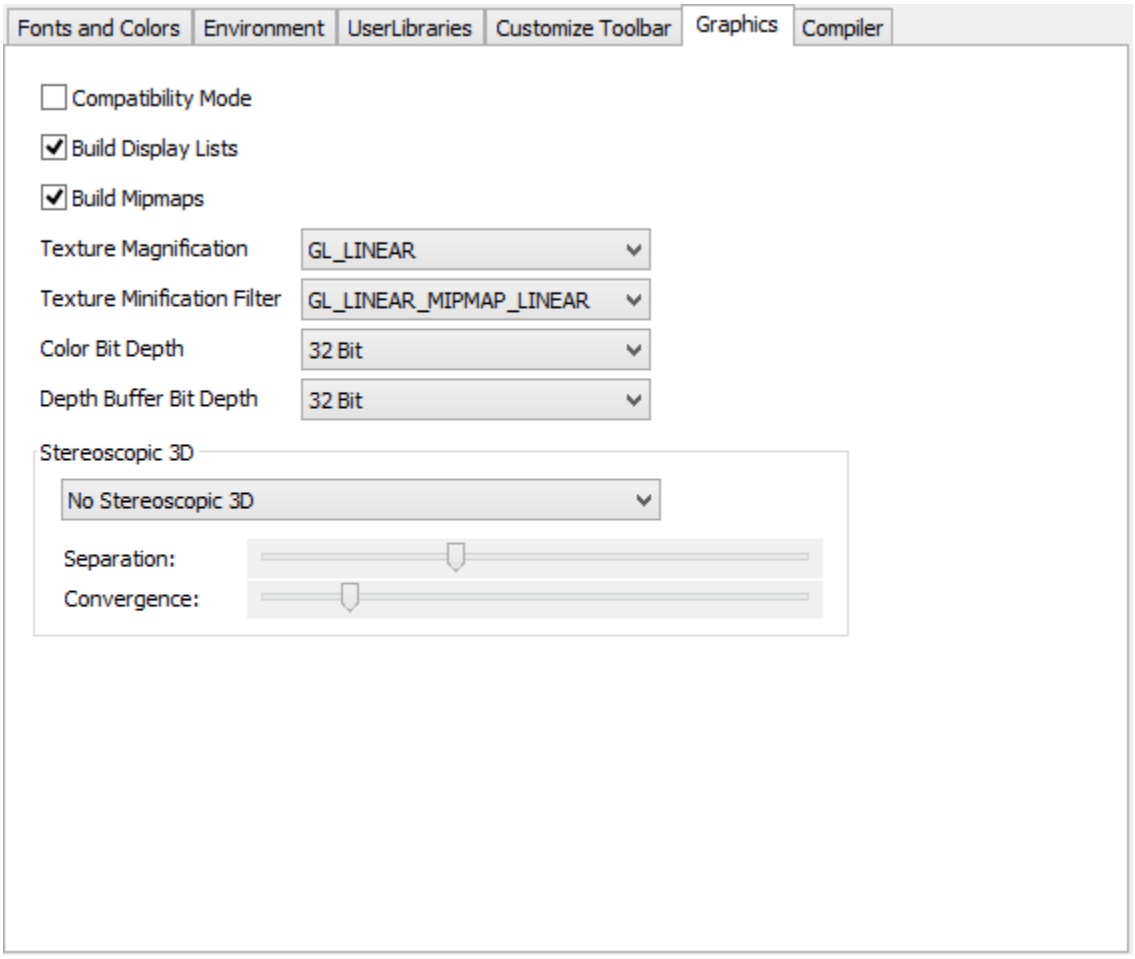
This tab page lets you specify a set of user libraries to be loaded when FlexSim starts up. The paths specified are relative to your Flexsim7/libraries directory.

Customize Toolbar



In this tab page you can customize which menu commands are accessible easily through the customizable section of the top toolbar.

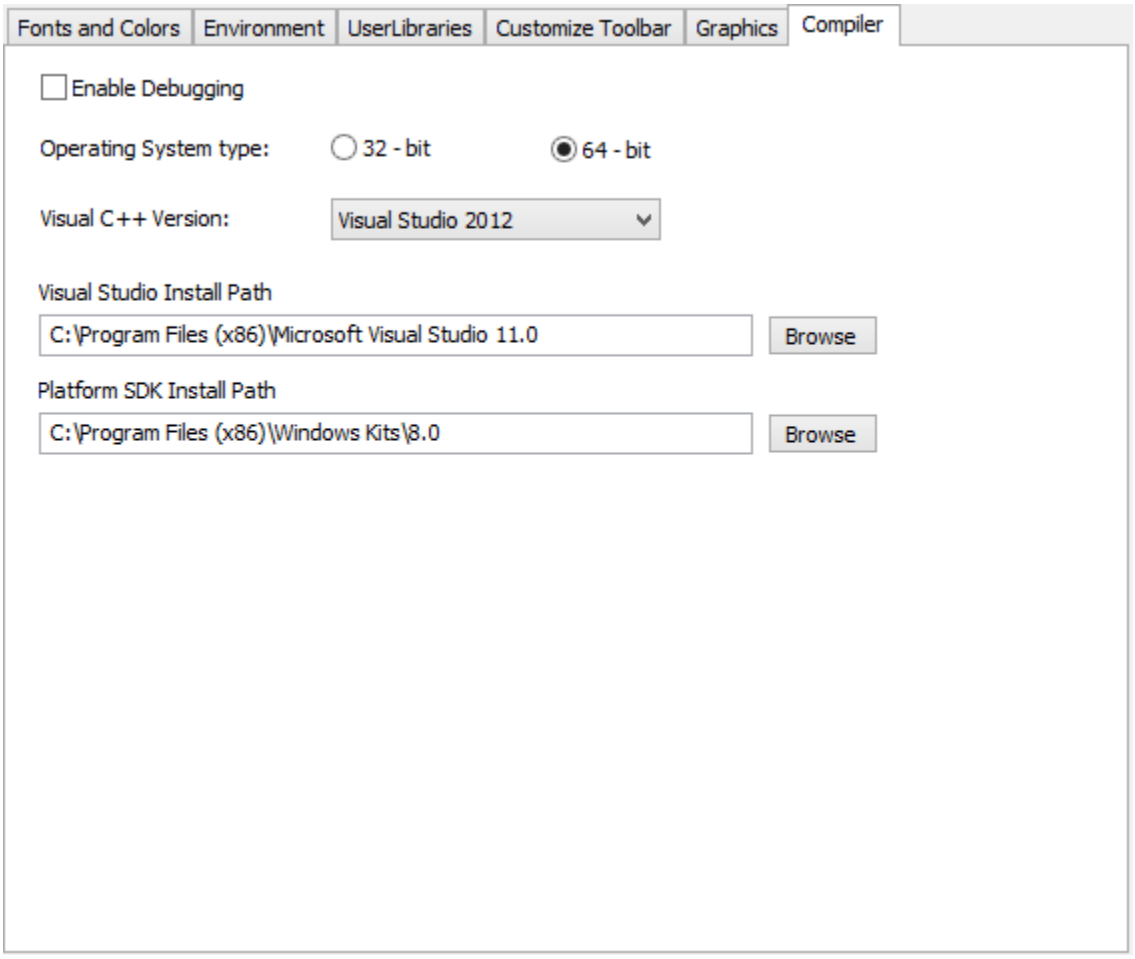
Graphics



This tab is used for customizing graphics settings so that FlexSim will run the best on your hardware.

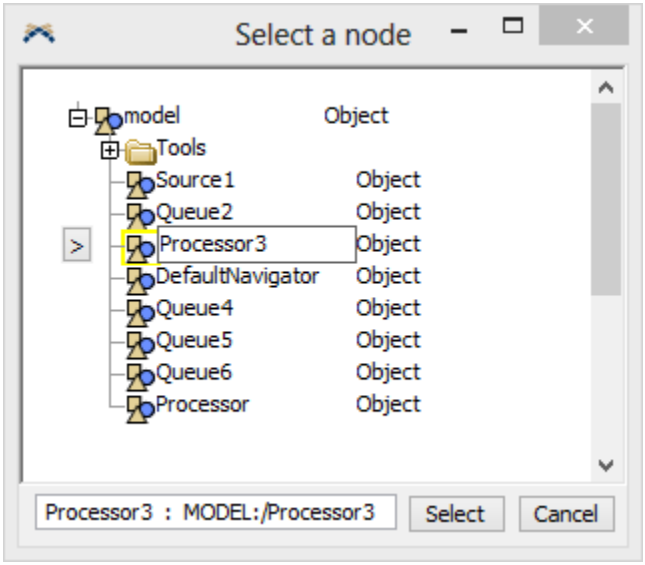
Note: If drag-dropping objects, making connections, or navigating the 3D view is not working properly, your graphics card may not be compatible with FlexSim. You should first try updating your graphics card driver from your computer's manufacturer. If that does not help, try updating the driver from the graphics card vendor (typically Nvidia or AMD). As a last resort if updated drivers do not resolve your issues, check the Compatibility Mode box to use your CPU for graphics processing.

Compiler



This tab is used for configuring Visual Studio to compile or debug C++ code.

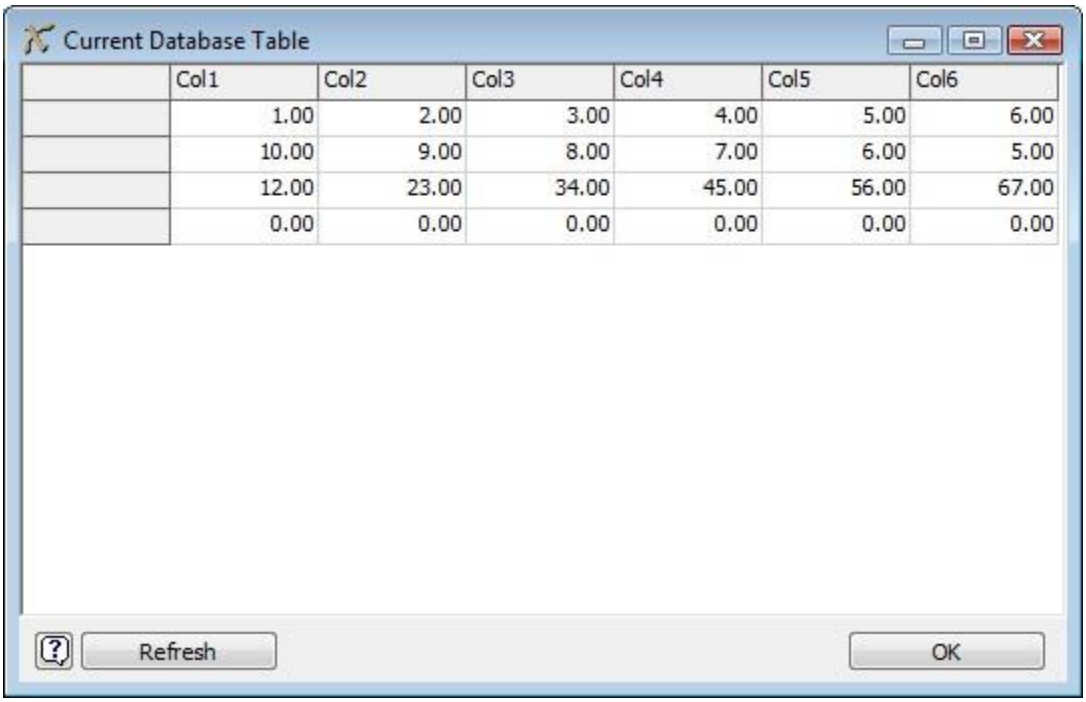
Tree Browse Dialog



This dialog allows you to browse for a node in the tree. The purpose of selecting a node is dependent on the context of the situation. Sometimes you will use this window to select a start node for a find/replace operation. Sometimes you will use this window to select an experiment variable in the Experimenter. You may also select objects or nodes in the Global Variables window.

Select the node you want by clicking on it in the tree view, then click on the Select button. The window will then close and return to the original calling window.

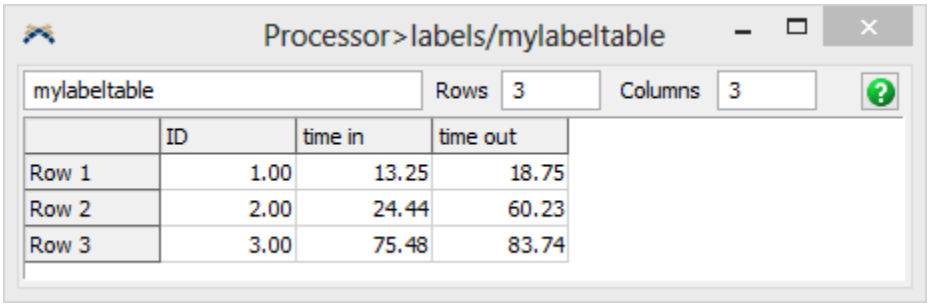
Database Table View



This table view is accessed from the View menu. It shows the currently active database table. Database tables are opened using the dbopen() command. For more information, refer to the command summary.

Table Editor

The table editor is used to edit a simple table in FlexSim, and can be opened by right-clicking on a node and choosing Explore > As Table.



Name - This is the table's name. It should be memorable and describe the table's function. The commands to read and write to them access them by name.

- Pins the entire table to a Dashboard as either a table of values, bar chart or line graph.

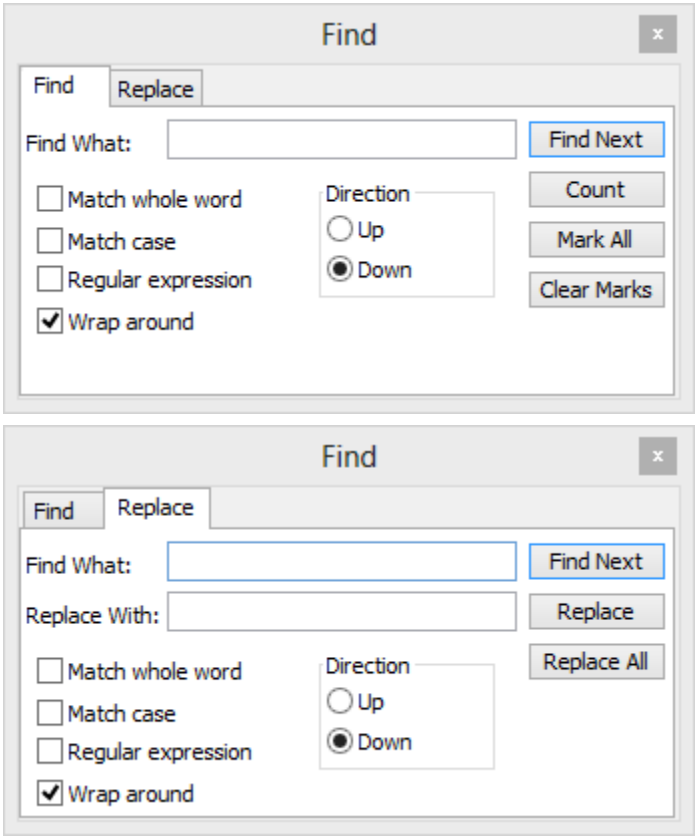
Rows - This is the number of rows in the table. After changing it, press Apply to update the table on-screen. Any new rows that were created can now be edited.

Columns - This is the number of columns in the table. After changing it, press Apply to update the table on-screen. Any new columns that were created can now be edited.

Editing the Table

To edit a cell in the table, click on the cell and type the data in the cell. Press the arrow keys to navigate between cells. Cells hold numbers by default, but can be set to hold string data by right-clicking on the cell and selecting Assign String Data.

Find and Replace



The Find/Replace window is available by holding the Ctrl key and pressing F. This window allows you to search the currently active window or view for the specified text and replace that text if desired.

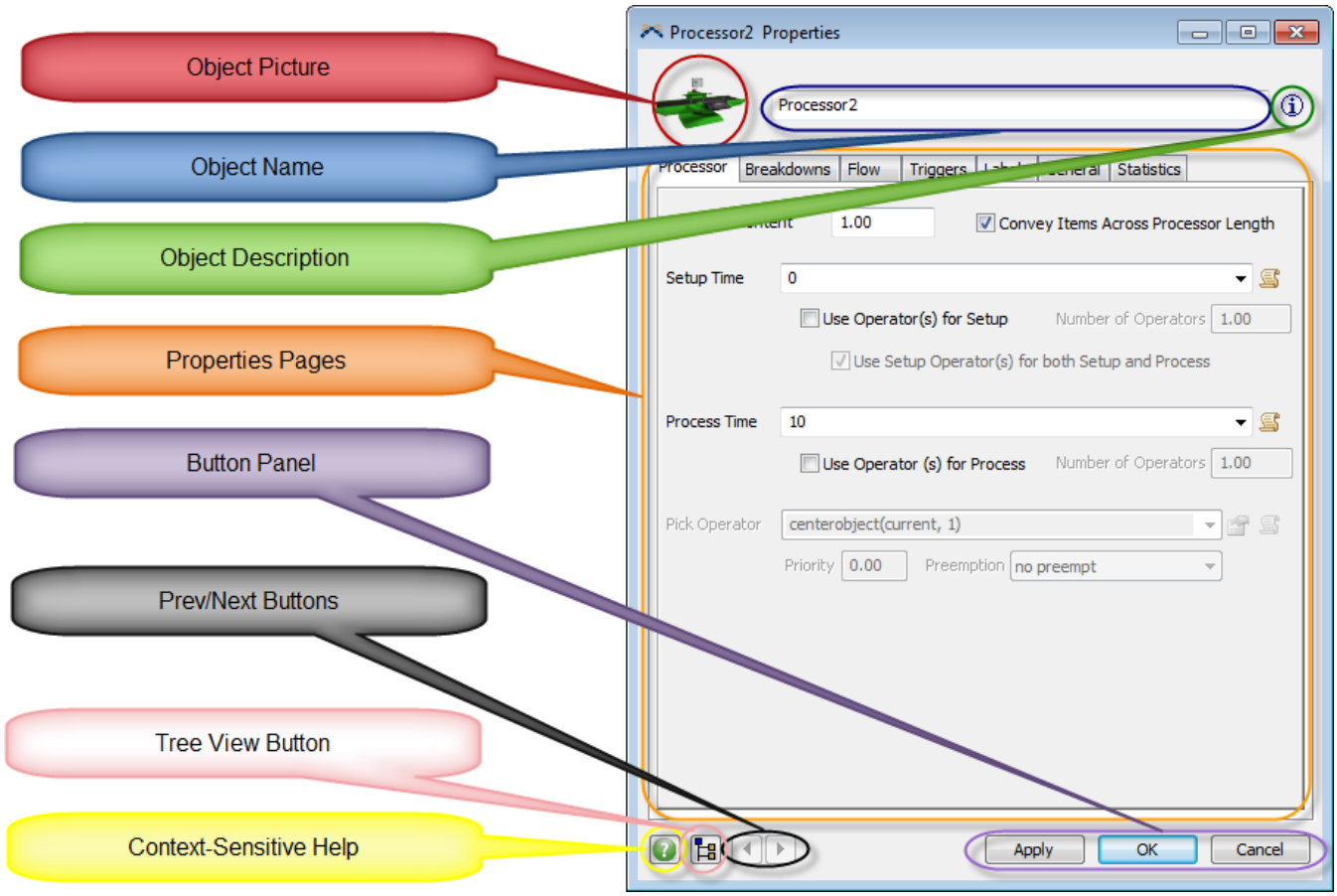
Object Windows

1. Overview
2. FixedResource Properties Pages
3. TaskExecuter Properties Pages
4. Fluid Properties Pages
5. Shared Properties Pages
6. Other Properties Pages

Object Properties Windows Overview

The Properties window allows you to configure attributes that are dependent on the type of object you are editing. For example, a Queue has a maximum content attribute, whereas a Combiner has a component list for receiving flowitems. The Properties window of the Queue is therefore different from the Properties window of the Combiner. However, there are often commonalities between Properties windows. For example, both the Queue and the Combiner can have a send-to strategy. Objects that have commonalities will usually contain portions of their Properties windows that are the same so that you can more quickly learn how to use these windows.

To bring up an object's Properties page, double click on the object or right click on the object and select the "Properties" option from the popup menu.



Object Picture

This picture shows you what type of object you are editing.

Object Name

Here you can give the object a different name. In specifying the name of the object, do not use any special characters like >,<,*,-, (,), etc. This may cause Flexsim to not behave correctly. Spaces and underscores should be the only non-alpha-numeric characters used. Also, do not begin the name with a number. You must press Apply or OK for this change to be applied. It is advisable to give each object in your model a unique name for reporting and other purposes. An error message will appear when you press Apply if another object has the same name as name you entered for this object.

User Description

This button opens an edit field where you can type a description or notes about the object. This description is saved with the object and can be viewed at a later time the same way it is edited: by pressing this button. Click anywhere in the window for the User Description edit field to apply and disappear.

Properties Pages

Properties windows are made up of a set of tab pages, depending on the object. Many pages are used by several objects. This reference provides documentation for each properties page. The properties pages are listed as follows. Within each help page are links to each object that uses that page.

Fixed Resource Pages - These pages are used by Fixed Resource objects.

- **BasicFR Advanced**
- **Breakdowns**
- **Combiner**
- **Conveyor**
- **Decision Points**
- **Flow**
- **Layout**
- **MergeSort Flow**
- **MultiProcessor**
- **Photo Eyes**
- **Processor**
- **ProcessTimes**
- **Queue**
- **Rack**
- **Separator**
- **Sink**
- **SizeTable**
- **Source**

Task Executer Pages - These pages are used by Task Executer objects.

- **ASRSvehicle**
- **BasicTE**
- **Breaks**
- **Collision**
- **Crane**
- **Dispatcher**
- **Geometry**
- **Robot**
- **TaskExecuter**
- **Transporter**

Fluid Pages - These pages are used by fluid objects.

- **Blender**
- **FluidConveyor**
- **FluidLevelDisplay**
- **FluidProcessor**
- **FluidToItem**
- **Generator**
- **Initial Product**
- **Inputs / Outputs**
- **ItemToFluid**
- **Layout**
- **Marks**
- **Mixer**
- **Percents**
- **Pipe**
- **Recipe**

- **Sensors**
- **Splitter**
- **Steps**
- **Tank**
- **Terminator**
- **Ticker**

Shared Pages - These pages are shared by multiple objects.

- **General**
- **Labels**
- **Triggers**

Other Pages

- **Container Functionality**
- **Display**
- **NetworkNode**
- **NetworkNodes**
- **Speeds**
- **Traffic Control**

Button Panel

Apply - Applies any changes made in the window to the object.

OK - Applies any changes made in the window to the object and closes the window.

Cancel - Closes the window without applying any changes made in the window. Some properties of the object are updated immediately when changing them in the window. These changes will still be applied and will not be undone when pressing Cancel. Examples of these immediately changing properties are Breakdowns, Labels, Appearance, Position, Rotation, Size, and Ports.

Prev/Next Buttons

These buttons will apply any changes made in the window to the object and change the focus of the window to the previous or next object of the same class type in the tree. This is helpful for quickly modifying properties of multiple objects with the same class type.

Tree View Button

This button opens a tree edit window showing just this object.

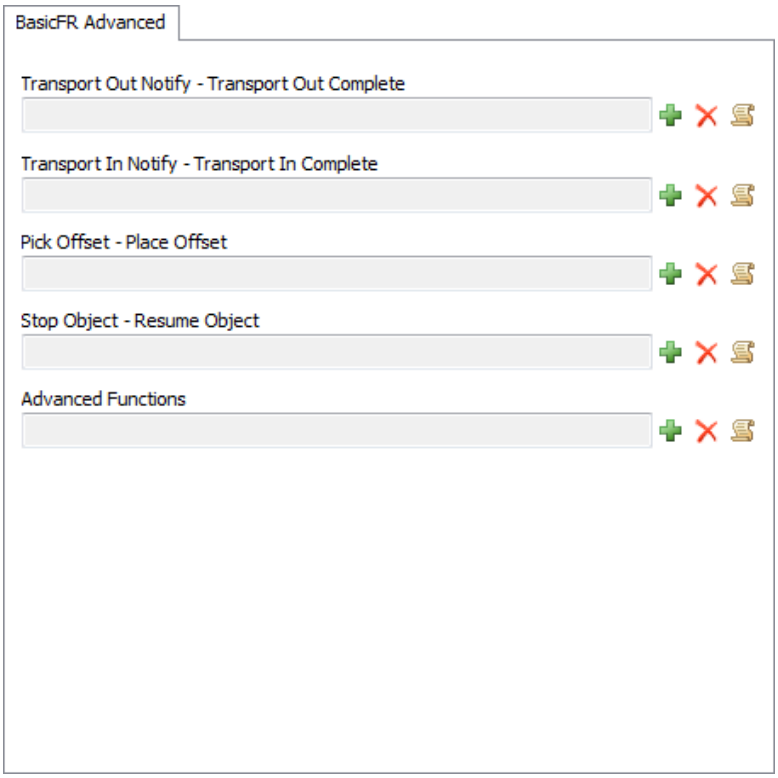
Context-Sensitive Help

This button will open the User's Manual to the page describing the currently selected tab.

FixedResource Properties Pages

1. BasicFR Advanced
2. Breakdowns
3. Combiner
4. Conveyor
5. Decision Points
6. Flow
7. Layout
8. MergeSort Flow
9. MultiProcessor
10. Photo Eyes
11. Processor
12. ProcessTimes
13. Queue
14. Rack
15. Separator
16. Sink
17. SizeTable
18. Source

BasicFR Advanced Page


BasicFR Advanced
Transport Out Notify - Transport Out Complete
Transport In Notify - Transport In Complete
Pick Offset - Place Offset
Stop Object - Resume Object
Advanced Functions

Transport Out Notify - Transport Out Complete - This trigger is fired at two different times. First is when the object is notified that a flowitem is going to exit the object using a transport. This function is referred to as transport out notify. The second time it is fired is when the transport has arrived, finished its load time, and is about to move the flowitem out. This is called transport out complete. A variable is passed into this function telling you which operation is applicable. In this field you can manage things like the nroftransportsout variable, as well as how to screen further output/input to the object.

Access variables for the transport out notify - transport out complete function are as follows:

current: The current object.

notifyoperation: This variable is 1 or 0. 1 means it is a transport out notify operation, 0 means it is a transport out complete operation.

item: This is a reference to the item that is going to leave this object

port: This is the output port number through which the item will exit

transporter: For a transport out complete operation, this is a reference to the transporter that is picking the item up.

nroftransportsoutnode: This is a reference to the object's nroftransportsout variable. The value of this node should be incremented by 1 at a notify operation, and decremented by 1 at a complete operation. You can also query the value of this node to know how many items are waiting for a transport to pick them up.

Transport In Notify - Transport In Complete - This trigger is fired at two different times. First is when the object is notified that a flowitem is going to enter the object using a transport. This function is referred to as transport in notify. The second time it is fired is when the transport has arrived, finished its unload time, and is about to move the flowitem into the object. This is called transport in complete. A variable is passed into this function telling you which operation is applicable. In this field you can manage things like the nroftransportsin variable, as well as how to screen further output/input to the object. For example, you may want to allow more than one flowitem to be in transit to the object at the same time. In such a case, you could call receiveitem when you are notified of an upcoming entry within this field.

Access variables for the transport in notify - transport in complete function are as follows:

current: The current object.

notifyoperation: This variable is 1 or 0. 1 means it is a transport in notify operation, 0 means it is a transport in complete operation.

item: This is a reference to the item that is going to enter this object

port: This is the input port number through which the item will enter

transporter: For a transport in complete operation, this is a reference to the transporter that is dropping the item off.

nroftransportsinnode: This is a reference to the object's nroftransportsin variable. The value of this node should be incremented by 1 at a notify operation, and decremented by 1 at a complete operation. You can also query the value of this node to know how many items are still in transit to this object.

Note on the transport out/in complete function return value: In the notify and complete operations of both transport in and transport out, if the function returns a value of 0, the object will assume that nothing was done and will execute its own default logic. If this function returns a 1, the object will assume the proper variable management was done and will do nothing. If this function returns a -1 and the operation is a complete operation, the object will again assume proper variable management, but in addition, it will notify the transporter that it is not ready to receive the flowitem. The transporter then must wait until it is notified that it can resume its operation. The reason you may need to use this is in case this object has been stopped using stopobject(). If so, you may not want any flowitems coming in or going out. If this is the case, then you will need to save off a reference to the transporter using the savestoppedtransportin() or savestoppedtransportout() function, and then return -1. Then, when it is ok for the transporter to resume its operation (usually from this object's resume object function) you will need to call resumetransportsin() and resumetransportsout() to notify all stopped transports that they may resume their operation.

Pick Offset - Place Offset - This trigger is fired at two different times. First is when a transport object is attempting to place, or unload, a flowitem into this object. This is called place offset. It should return an offset location for the transport to offset to before placing the item. The second time this is called is when a transport is about to pick, or load, an item from this object. This is called pick offset. It should again return an offset location for the transport to offset to before picking the product up.

Access variables for the pick offset - place offset function are as follows:

current: The current object.

pickoperation: This variable is 1 or 0. 1 means it is a pick operation, 0 means it is a place operation.

item: This is a reference to the item that is being picked or placed

otherobject: This is a reference to the object that is picking or placing the item

xvalnode, yvalnode, zvalnode: These parameters are references to nodes whose values should be set in this function, and will represent the offset location returned by this function. For example, if I want the picking/placing object to offset 10 in the x direction, 0 in the y direction, and 5 in the z direction, I would set xvalnode to 10 using the setnodenum() command, yvalnode to 0, and zvalnode to 5.

Note on the pick/place offset return value: If you are implementing your own pick/place logic here, and do not want the default logic to be executed, you must return 1 from this function. If the object gets a return value of 0 from this function, it will assume that nothing was done, and will execute its own default logic.

Stop Object - Resume Object - This trigger is fired at two different times. First is when the stopobject() command is called for this object. Second is when the resumeobject command is called for this object. This field should define a strategy for how the object will "stop", and how it will "resume". This field should also manage data for remembering how many stop requests have been made, what the state of the object was before it was stopped, etc.

Access variables for the stop object - resume object function are as follows:

current: The current object.

stopoperation: This value is either a 1 or 0. 1 means the stopobject() command is being called on this object. 0 means the resumeobject() command is being called on this object.

stopstate: This value is only applicable for a stop operation. It specifies the requested state passed into the stopobject() command.

nrofstopsnode: This is a reference to this object's nrofstops variable. If this is a stop operation, the value of this node should be incremented by 1 within this function. If it is a resume operation, the value of this node should be decremented by 1 within this function. Also, you will get the value of this node to know how many stopobject() commands have been called on this object. When the object is stopped for the first time (the value of the node goes from 0 to 1), you should execute logic specifying how to stop this object. When the object is resumed for the final time (the value of the node goes from 1 to 0), you should execute logic specifying how to resume this object.

timeoflaststopnode: This is a reference to this object's timeoflaststop variable. When this object is stopped for the first time, this node should be set to the current time, so you can know the amount of time the object was stopped when it is finally resumed.

statebeforestopnode: This is a reference to this object's statebeforestopped variable. When this object is stopped for the first time, this node should be set to the object's current state, so that you can know which state to go back to when the object is resumed.

Note on the stop/resume object return value: If you are implementing your own stop/resume logic here, and do not want the default logic to be executed, you must return 1 from this function. If the object gets a return value of 0 from this function, it will assume that nothing was done, and will execute its own default logic.

Advanced Functions - This trigger is fired for several different notifications or functions that are called on the object. For the most part, you will not need to implement any logic for these notifications, but they are nonetheless made accessible to you. The return value of this function should either be a 1 or a 0. If 0, the object will execute the default functionality associated with the given notification. If 1, the object not do any default functionality, but assumes that the function has overridden the default.

The type of notification called for the advanced function is passed in as parval(1), or msgtype. This parameter can be one of several values. These values are listed as follows:

ADV_FUNC_CLASSTYPE: This is a request to get the type of class of the object. The classtype should be returned as an integer with certain bits set high. You can construct this value using the bitwise OR operator | and several classtype macros. For example, a FixedResource's classtype is: CLASSTYPE_FLEXSIMOBJECT > CLASSTYPE_FIXEDRESOURCE.

ADV_FUNC_DRAGCONNECTION: This function is called when a keyboard key is held down, and the user clicks and drags from this object to another object. In this case, the object to which the mouse was dragged is passed in as parnode(2), the ascii value of the key that was clicked is passed in as parval(3), and the classtype value of the object is passed in as parval(4).

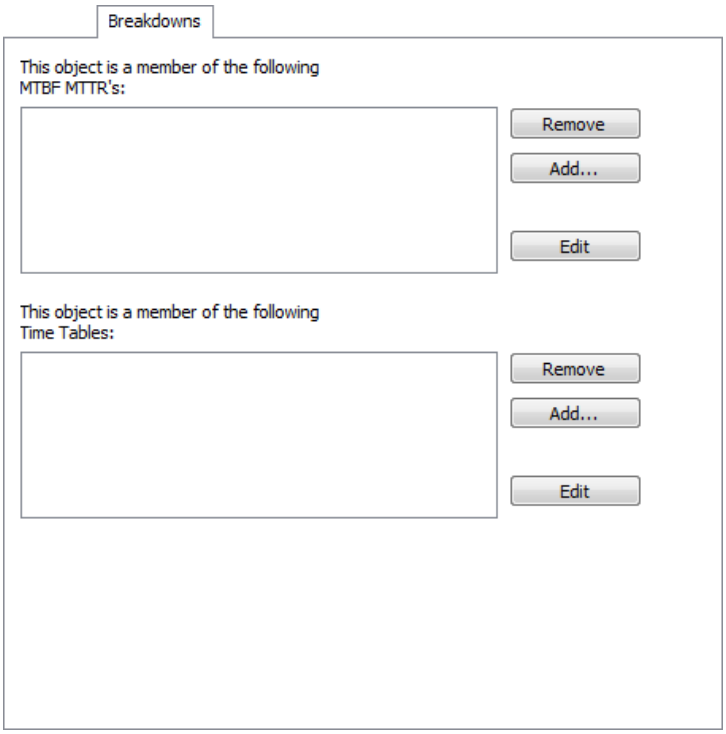
ADV_FUNC_CLICK: This function is called when the object is clicked on. Here a reference to the view in which it was clicked is passed in as parnode(2), and the click code is passed in as parval(3). Possible click codes are: DOUBLE_CLICK, LEFT_PRESS, LEFT_RELEASE, RIGHT_PRESS, RIGHT_RELEASE.

ADV_FUNC_KEYEDCLICK: This function is called when a key on the keyboard is held down and the object is clicked on. Here the view is passing as parnode(2), the click code is passed in as parval(3), and the ascii value of the pressed key is passed in as parval(4). Possible click codes are: DOUBLE_CLICK, LEFT_PRESS, LEFT_RELEASE, RIGHT_PRESS, RIGHT_RELEASE.

This Page is Used By

BasicFR

Breakdowns Page



MTBF MTTR Member List - This is a list of all the MTBF MTTR objects that have this object as one of its members. Each MTBF MTTR object can be connected to more than one object in the model. And each object can be controlled by more than one MTBF MTTR object. For more information about MTBF MTTR objects, refer to the Modeling Tools section about MTBF/MTTR objects.

Remove - This button removes the object from the selected MTBF MTTR object's member list.

Add.. - This button opens a listbox of all the MTBF MTTR objects in the model. You can select an MTBF MTTR object from the list to add this object to that MTBF MTTR object. You can also select "Add New MTBF MTTR" to create a new MTBF MTTR object in your model and add this object to that MTBF MTTR object.

Edit - This button allows you to edit the MTBF MTTR object's properties, including the timing of the breakdowns and repairs. For more information about MTBF MTTR objects, refer to the Modeling Tools section about MTBF/MTTR objects.

Time Tables Member List - This is a list of all the Time Table objects that have this object as one of its members. Each Time Table object can be connected to more than one object in the model. And each object can be controlled by more than one Time Table object. For more information about Time Table objects, refer to the Modeling Tools section about Time Tables.

Remove - This button removes the object from the selected Time Table object's member list.

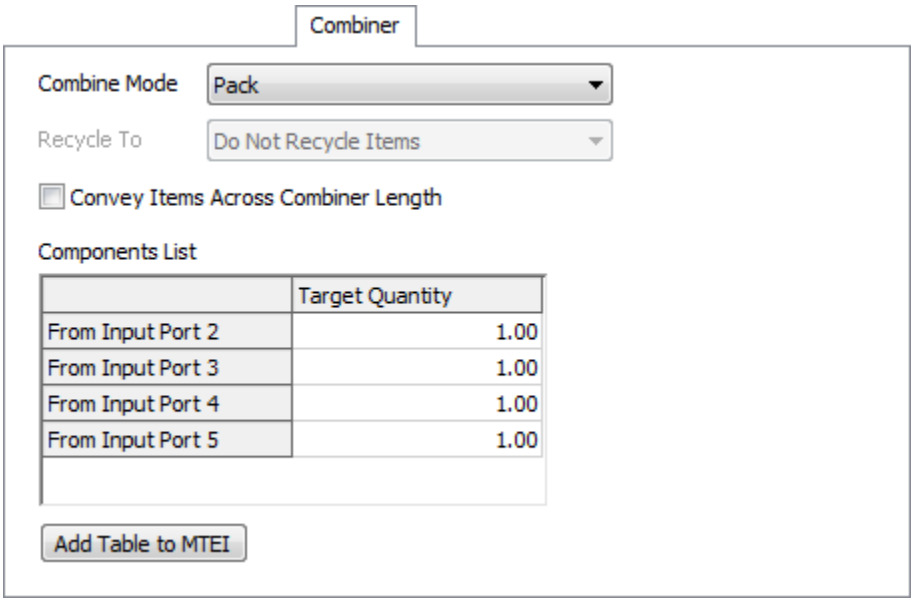
Add.. - This button opens a listbox of all the Time Table objects in the model. You can select an Time Table object from the list to add this object to that Time Table object. You can also select "Add New Time Table" to create a new Time Table object in your model and add this object to that Time Table object.

Edit - This button allows you to edit the Time Table object's properties, including the timing and duration of scheduled breakdowns. For more information about Time Table objects, refer to the Modeling Tools section about Time Tables.

This Page is Used By

Processor
Combiner
Separator

Combiner Page



Pack/Join/Batch - Selects the mode that the Combiner is operating in.

Recycle To (Join mode only) - In join mode, the objects that come in through ports greater than 1 are destroyed after the combiner is finished processing. Rather than destroying the extra flowitems, you can use this option to recycle them to a specific flowitem recycling bin. To learn more about recycling, refer to the Sink Page.

Convey Items Across Combiner Length - If checked, flowitems will travel across the Combiner during their process time.

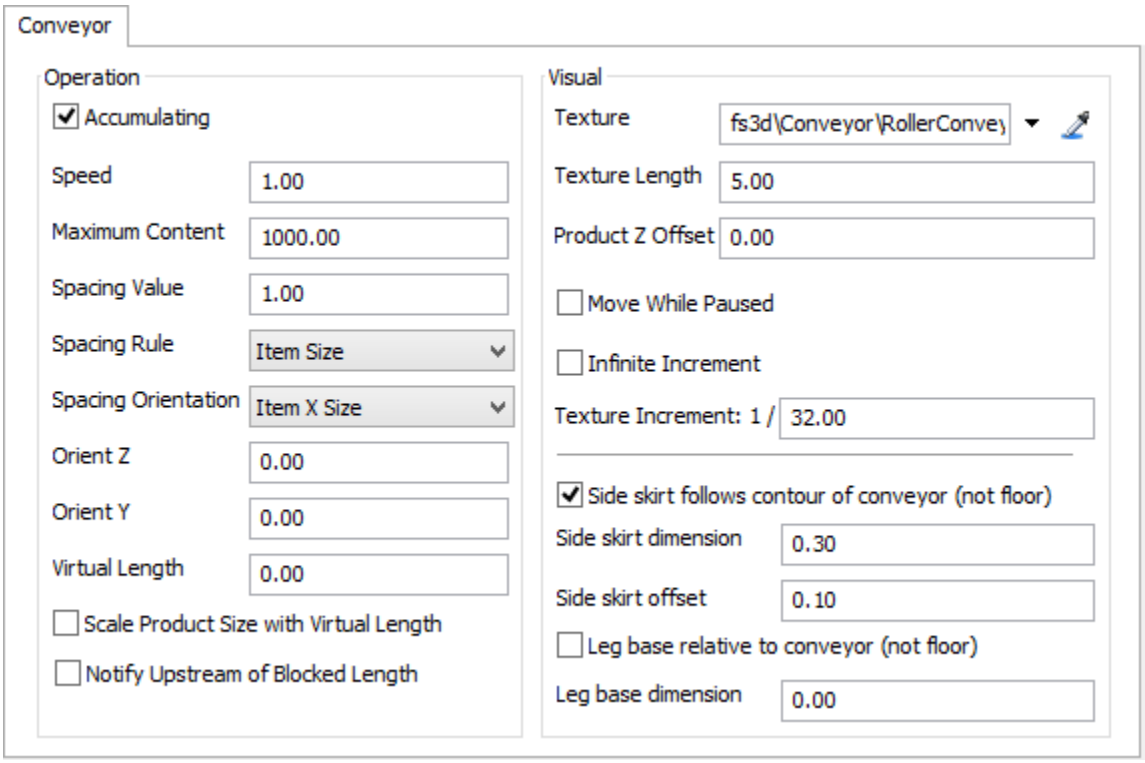
Components list - This table is used to define how many of each type of flowitem the combiner will collect before sending the completed collection downstream. The combiner will use the flowitem that arrives through input port one as the container object and will only accept one of them. Each row in the table represents arrivals from input ports numbered two and above. If you make additional connections while this window is open, you will need to close the Properties window and reopen it in order for your changes to register. To update this list dynamically during a model run, use the Update Combiner Component List picklist option in the OnEntry trigger.

TargetQuantity - The required number of flowitems to be received through the associated input port for each operation.

This Page is Used By

Combiner

Conveyor Page



Operation

These properties define how the conveyor functions.

Accumulating - If this box is not checked, the conveyor is non-accumulating. In a non-accumulating conveyor, the entire conveyor stops if a flowitem reaches the end and cannot exit. If accumulating, the flowitems will continue traveling the length of the conveyor until they run into a stopped flowitem or reach the end. The distance between accumulated flowitems is determined by the spacing rule.

Speed - This number defines the speed that flowitems travel at as they move down the conveyor.

Maximum Content - This number sets a limit on how many flowitems can be on the conveyor at one time. The maximum number of flowitems on a conveyor is usually determined by the length of the conveyor and the size of the flowitems, but this allows you to screen it even more.

Spacing Value - A flowitem will stop when its front edge is a certain distance away from the front edge of the flowitem in front of it. That distance is defined by the spacing value. This number is used in different ways, as you define in the Spacing Rule.

Spacing Rule - This defines how much space the conveyor leaves between flowitems that are on it.

- **Item Size -** A flowitem will stop moving when it reaches the flowitem in front of it.
- **Item Size + Spacing Value -** A flowitem will stop when it is a certain distance from the back of the flowitem in front of it. That distance is defined by the spacing value.
- **Spacing Value -** A flowitem will stop when it is a certain distance from the front of the flowitem in front of it.
- **Item Size * Spacing Value -** A flowitem will stop when it is a certain distance from the back of the flowitem in front of it. That distance is defined by the preceding item's size times the spacing value.

Spacing Orientation - If the Spacing Rule includes the item's size, this field determines which dimension of an item defines its size on the conveyor. Options are x, y, or z size.

Orient Z, Orient Y - These fields are for visual purposes and do not affect the operation of the conveyor. They are used in conjunction with the Spacing Orientation field to orient the flow items correctly. They define a rotation in degrees around the z and y axes of the conveyor for flow items to be rotated. For

example, if in the Spacing Orientation field Item Y Size is chosen instead of the default Item X Size, you will want to specify the Orient Z field as either 90 or 270, so that the y dimension of the flow item aligns with the length of the conveyor, instead of the x size. If Item Z Size is chosen, then you will want the Orient Y field to be either 90 or 270.

Virtual Length - This field allows you to define a specific length for the conveyor, different from the actual length of its layout. If 0 is input (default), the conveyor's normal layout length will be used. Otherwise, the value in this field will be used as the conveyor's length. This field is used if you want to simulation a very long conveyor without having it traverse a huge distance in your model, or if you want to specify an exact distance.

Scale Product with Virtual Length - This is for visual purposes and does not affect the operation of the conveyor. If you have defined a very large Virtual Length, this field allows you to scale the size of the products that flow down the conveyor so that they don't overlap. For example, if the conveyor's layout length is 10, but you've specified a virtual length of 100, then items with a size of 1 would only take up 1/100th of the conveyor's virtual length. This is 0.1 units in the real layout, but since a flow item is 1 unit, it overlaps with other flow items. If this box is checked, then the size of the flow item will be shrunk to 0.1 units.

Notify Upstream of Blocked Length - This tells the conveyor to notify upstream conveyors when products on the conveyor extend past the conveyor's leading edge.

Visual

These properties define how the conveyor looks in the model. They do not affect the conveyor's operational functionality. These properties in combination give you unlimited flexibility in the look of your conveyor. Often configuring these settings is done by trial and error, but you should be able to learn quickly how each parameter affects the conveyor's look.

Texture - This file will be drawn on the conveyor to change its basic appearance. The texture is divided into three parts. The left third of the texture is drawn onto the left side skirt, the middle third is drawn on the top, and the right third is drawn on the right side skirt.

Texture Length - This number defines the length along the plane of the conveyor to stretch one copy of the texture before repeating the texture.

Product Z Offset - Changing this number will change how far above or below the conveyor the flowitems are drawn. A value of 0 puts the items directly on the conveyor. A negative value will put the items below the conveyor's plane.

Move While Paused - By default, the conveyor's texture animation is only updated as the simulation runs. If this box is checked, the conveyor will animate with every refresh of the 3D window in addition to refreshing while the model is running.

Infinite Increment - Checking this box makes it so that the conveyor's texture is animated smoothly instead of at fixed increments. You would use this option with a texture that moves along the conveyor, such as a belt texture.

Texture Increment - The conveyor's texture animation is updated at particular increments. For example, the roller texture has 32 rollers in its texture. The texture is only animated by shifting the texture by the specified value. This makes the animation look like individual rolling wheels instead of a flat rolling texture. You can set the Texture Increment to 1 to prevent the texture from animating.

Side skirt follows contour of conveyor (not floor) - If this box is checked, the side of the conveyor will be drawn starting a certain distance away from the conveyor. If it is not checked, it is drawn starting a certain distance away from the floor (the z location of the conveyor). Either way, the sides connect to the conveyor’s contour.

Side Skirt Dimension - This value is used to determine how tall the skirt is.

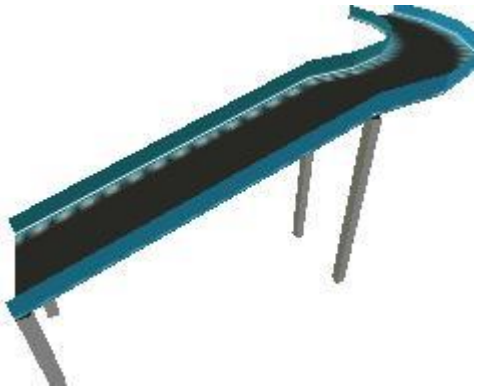
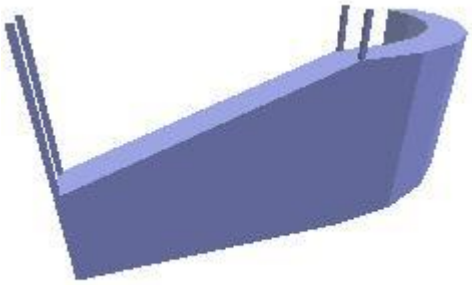
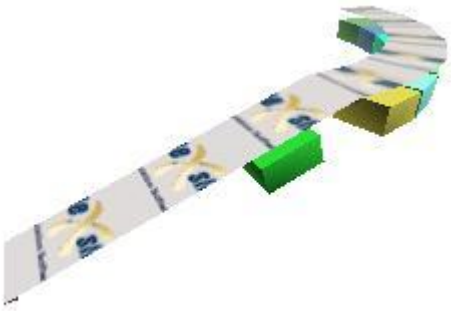
Side Skirt Offset - This value is used to offset where the side skirt is drawn.

Leg base relative to conveyor - If this box is checked, the legs of the conveyor will be drawn starting a certain distance away from the conveyor. If it is not checked, they are drawn starting a certain distance away from the floor (the z location of the conveyor). Either way, the sides connect to the conveyor’s contour.

Leg base dimension - This value is used to determine where the legs of the conveyor begin.

Visual Examples

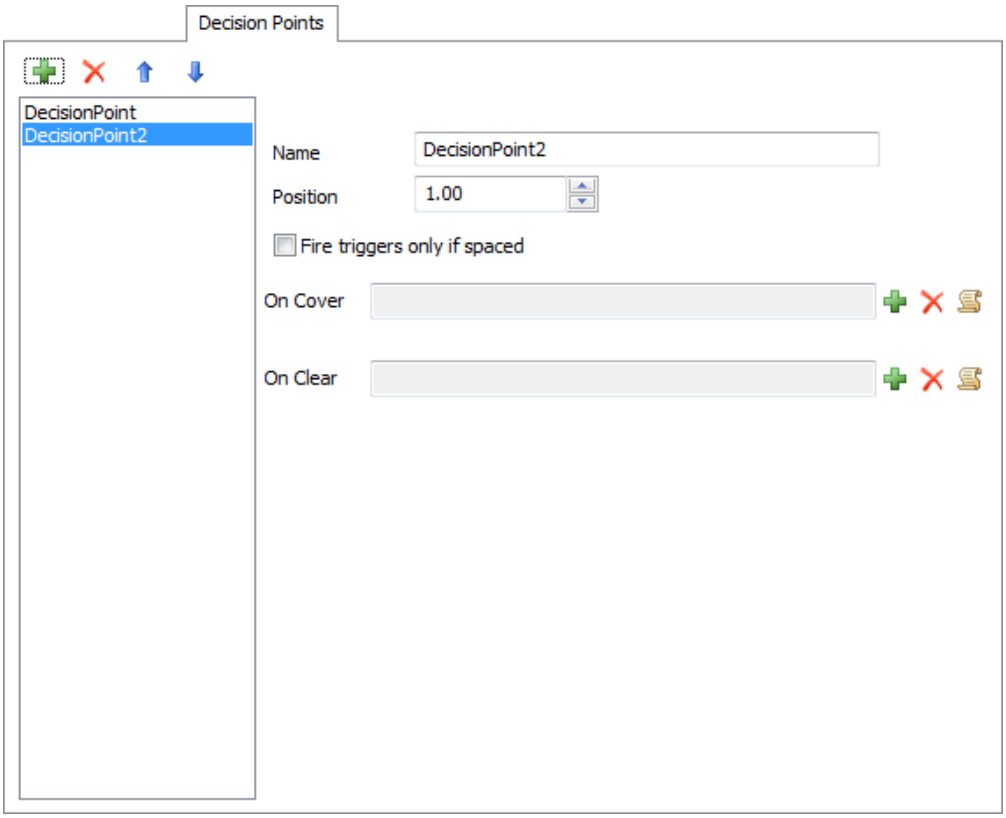
Here are some examples of different looks you can give your conveyor. They might not be completely practical, but they are there to show the flexibility and give you ideas. Attached with each picture is a small explanation of what was changed from the default visualization of the conveyor.

<table>
<tr>
<td> 
Belt conveyor with side rail
Side Skirt Dimension = -0.2
Texture = fs3d/BeltConveyor.bmp</td>
<td> 
Blue hanging conveyor with big side skirt
Texture = <blank>
Skirt follows contour unchecked
Skirt dimension = 0
Leg base dimension = 2.5
color changed to blue</td>
</tr>
<tr>
<td> 
Hanging tape conveyor with Flexsim texture
Texture = flexsim.bmp
Product Z Offset = -0.5
Side Skirt Dimension = 0
Leg base relative checked
Leg base dimension = 0
Texture Length = 1</td>
<td>
Conveyor with only side skirts
Custom texture with transparency.
Product Z Offset = -0.5</td>
</tr>
</table>

This Page is Used By

Conveyor
MergeSort
BasicConveyor

Decision Points Page



- Click this button to add a new DecisionPoint to the list below.

- Click this button to delete the selected DecisionPoint from the list below.

- Click this button move the selected decision points down in the list. This does not change the physical position of the decision point along the Conveyor, only it's ranking in the list.

- Click this button to move the selected decision points up in the list. This does not change the physical position of the decision point along the Conveyor, only it's ranking in the list.

The list contains each of the decision points for this basicconveyor. To edit a decision point, select it in the list and change its name and other details to the right.

Decision Point Details

For each decision point there will be a space to define the point with the following information to define the point:

Name - This field allows you to define the name of each point.

Position - This value specifies the position of the decision point relative to the start of the conveyor.

Fire triggers only if spaced - If this is checked, the OnCover and OnClear triggers will only fire if the flowitems are spaced apart.

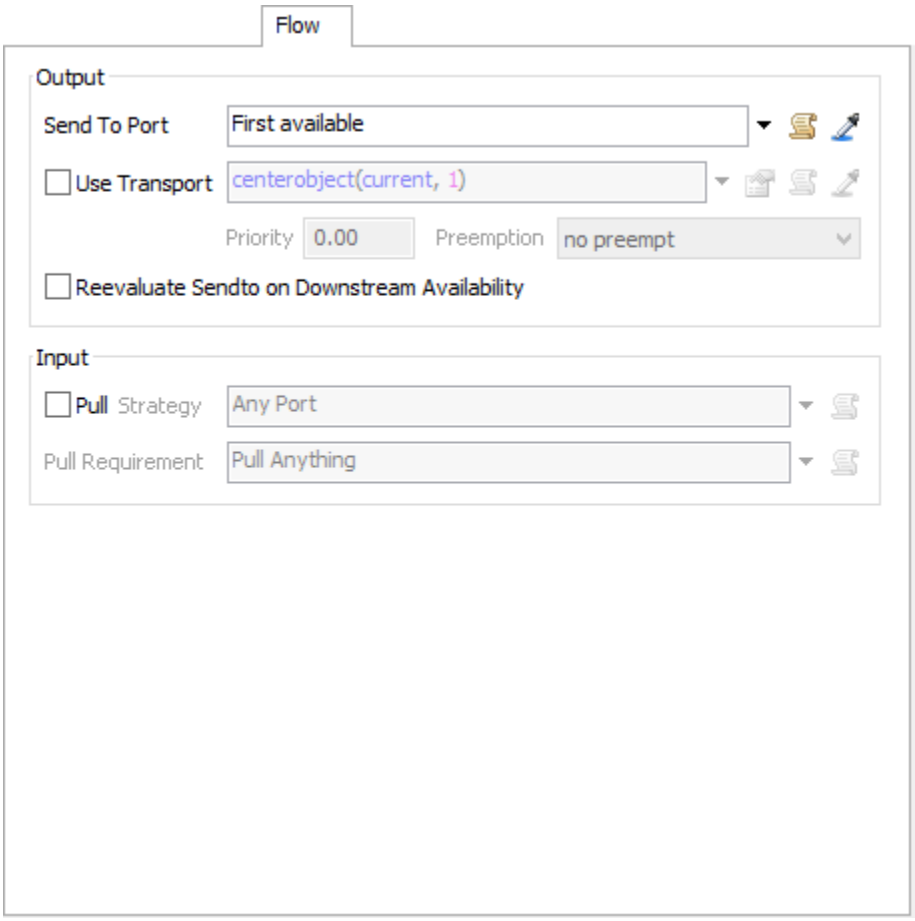
OnCover - This picklist fires when the decision point is first covered. See also On Cover picklist.

OnClear - This picklist fires as the flowitem clears the decision point. See also On Clear picklist.

This Page is Used By

BasicConveyor

Flow Page



For detailed information on this functionality, refer to the FixedResource.

Output

These properties determine how the object sends flowitems downstream.

Send To Port - This picklist returns the output port number connected to the object that the flowitem should be moved to. If 0 is returned, all outputs are opened and the flowitem is moved to the first downstream object that is able to receive it. See Send To Port picklist.

Use Transport - If this box is checked, the object will request a transport using the Request Transport From picklist to move the flowitem downstream. If it is not checked, the flowitem will be moved automatically.

Request Transport From - This picklist is only available if "use transport" is checked. This function returns a reference to the Dispatcher or Transporter that will be used to move the flowitem. See Transport Dispatcher picklist.

Priority - This parameter is only available if "use transport" is checked. This value sets the priority of the task sequence that will be sent to the transporter or dispatcher. Transporters and dispatchers generally sort task sequences so that sequences with higher priorities will be performed first. Task sequences with the same priority will be performed in the order that they were received.

Preempt - This parameter is only available if "use transport" is checked. If set to one of the preempting values, the task sequences sent to the transporter will automatically preempt whatever the transporter is doing at the time. This may cause the transporter to perform tasks that would normally not be allowed, such as carrying more flowitems than its capacity. For more information on preempting task sequences, see Task Sequence Preempting.

Reevaluate Sendto on Downstream Availability - If checked, the Send To Port will be re-evaluated every time a downstream object becomes available. It's important to note that this is only executed when

the downstream object becomes available. It does not continuously evaluate just because a downstream object is already available. If you want to manually force a re-evaluation at some other time than when a downstream object becomes available, then you can do so by calling the openoutput() command on this object.

Input

These properties define how an object pulls flowitems from upstream objects.

Pull - If this box is checked, the object will pull flowitems from upstream objects. The upstream objects should open all their output ports to allow the object to pull the flowitems it needs.

Pull Strategy - This parameter is only visible if "Pull" is checked. This picklist returns the input port number connected to the object that the next flowitem is to be pulled from. This field is evaluated only on reset of the model and when the pulling object becomes ready to receive its next flowitem. For a Processor with a capacity of 1, this means that the Pull Strategy field will only be evaluated once right after each flowitem exits the Processor. For a Conveyor, this field will be evaluated after a flowitem enters the conveyor and travels its product length. Opening and closing ports does not trigger this field to re-evaluate. See Receive From Port picklist.

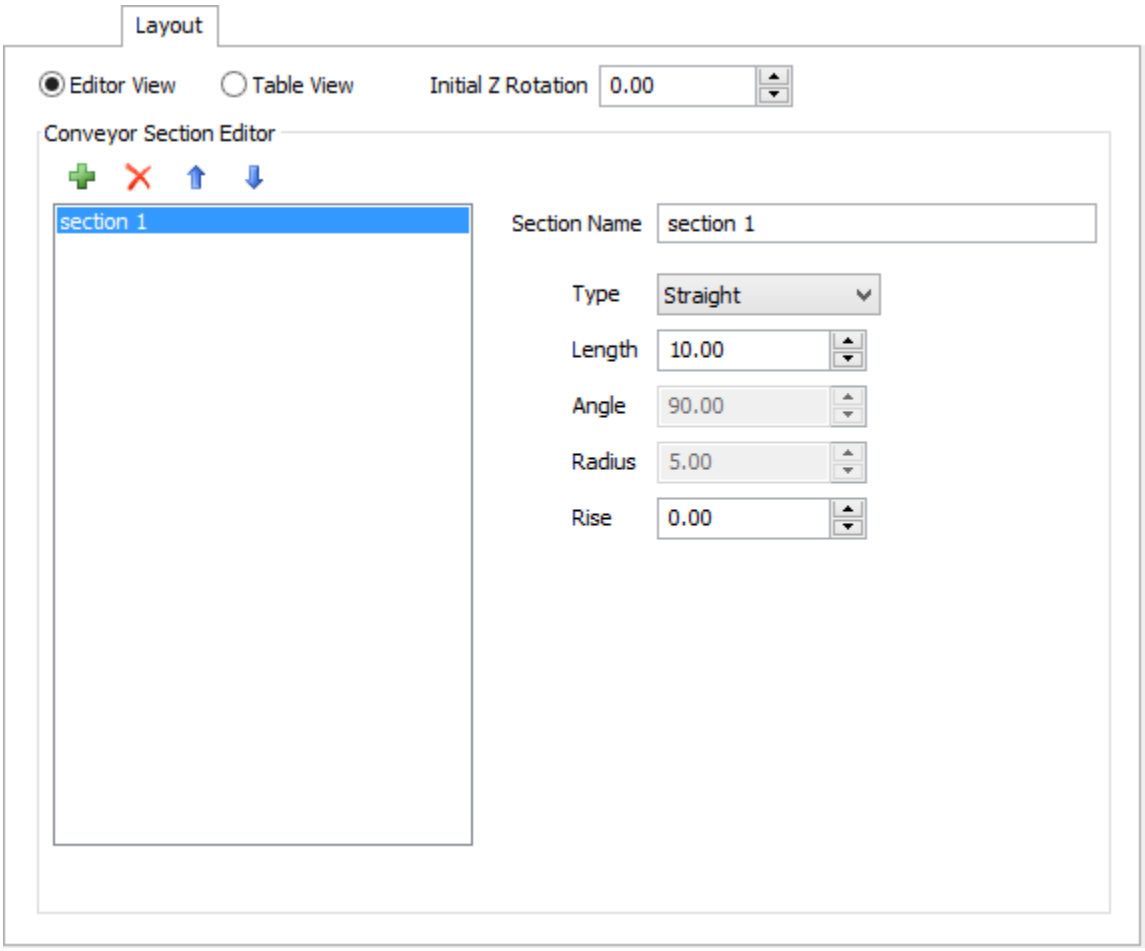
Pull requirement - This parameter is only visible if "Pull" is checked. This picklist needs to return either a true or a false (1 or 0). This field is evaluated when considering whether or not to pull in a particular flowitem from the upstream object that was defined by the "Pull from port" field. This field will only be evaluated for flowitems that are in the "ready" state (i.e. FRSTATE_READY) meaning the flowitems are ready to leave the upstream object. Basically, the "Pull Requirement" field is evaluated for every "ready" flowitem immediately after the "Pull from port" field gets evaluated. The field is evaluated again for each new flowitem that later becomes ready in the upstream object. See Pull Requirement picklist.

Reevaluate Pull Requirement on All Items When Each Upstream Item is Released - This parameter is only visible if "Pull" is checked. If checked, the object will re-evaluate the pull requirement for all released flowitems upstream every time a new flow item is released. This is much like the Reevaluates Sendto on Downstream Availability check box, in that you may need to explicitly call openinput() if you want to manually trigger the re-evaluation of the pull requirement.

This Page is Used By

Source
Queue
Processor
Combiner
Separator
MultiProcessor
Conveyor
Rack
MergeSort
BasicConveyor

Layout Page



Editor View - Displays the editor view GUI. (shown above)

Table View - Displays the table view GUI. (see below)

Initial Z Rotation - This number sets the rotation of the conveyor, or the direction of the first section of the conveyor. This is the same as editing the Z rotation in the Properties window.

- This button adds a straight section to the conveyor after the currently selected row. It adds the section to the end of the conveyor if nothing is selected in the table.

- This button deletes the currently selected row of the section table. It deletes the last row if nothing is selected in the table.

- Moves the selected section of conveyor up in the list, effectively moving the section of the conveyor.

- Moves the selected section of conveyor down in the list, effectively moving the section of the conveyor.

Section Name - The name of the section. This is purely for the modeler's convenience is identifying sections of the conveyor and has no impact on the model.

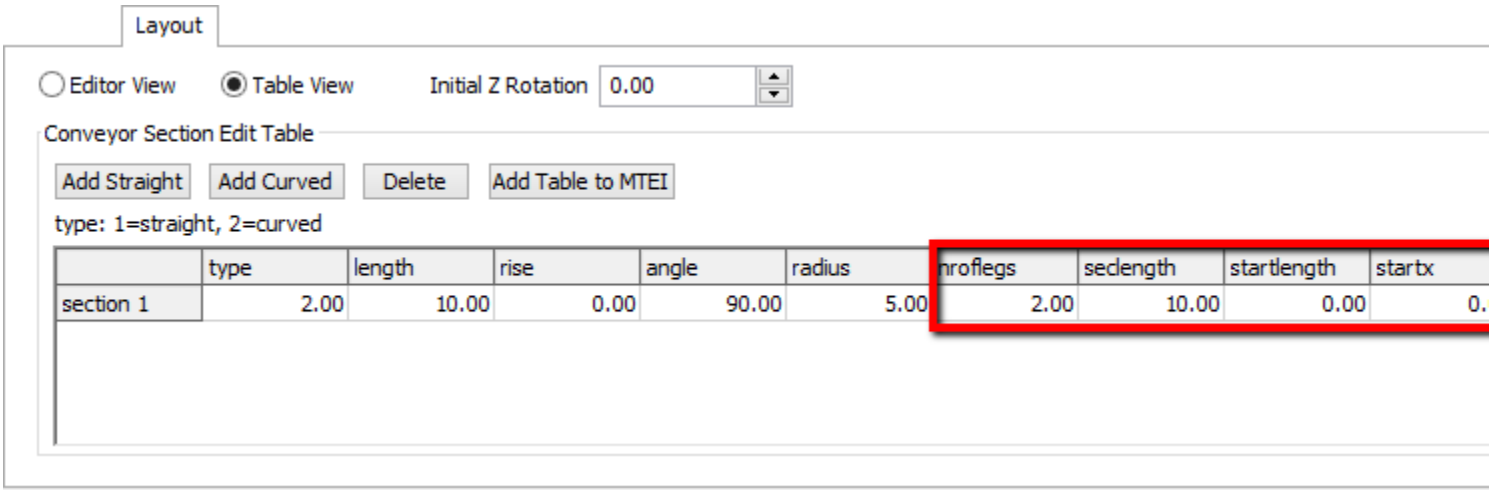
Type - This list lets you choose whether the currently selected section is curved or straight.

Length - This number defines how long the section is if it is a straight section. The value is ignored if it is a curved section.

Angle - This number defines the angle in degrees that the segment will turn if it is curved. It can be positive or negative and can be greater than 360. This value is ignored if the section is a straight section.

Radius - This number defines the radius of the turn in the curved section. The radius is measured from the center of the turn to the center of the conveyor. This value is ignored if the section is a straight section.

Rise - This value defines the difference in height between the start of the segment and the end. For example, if this value is 3, the segment will end 3 units higher than it began.



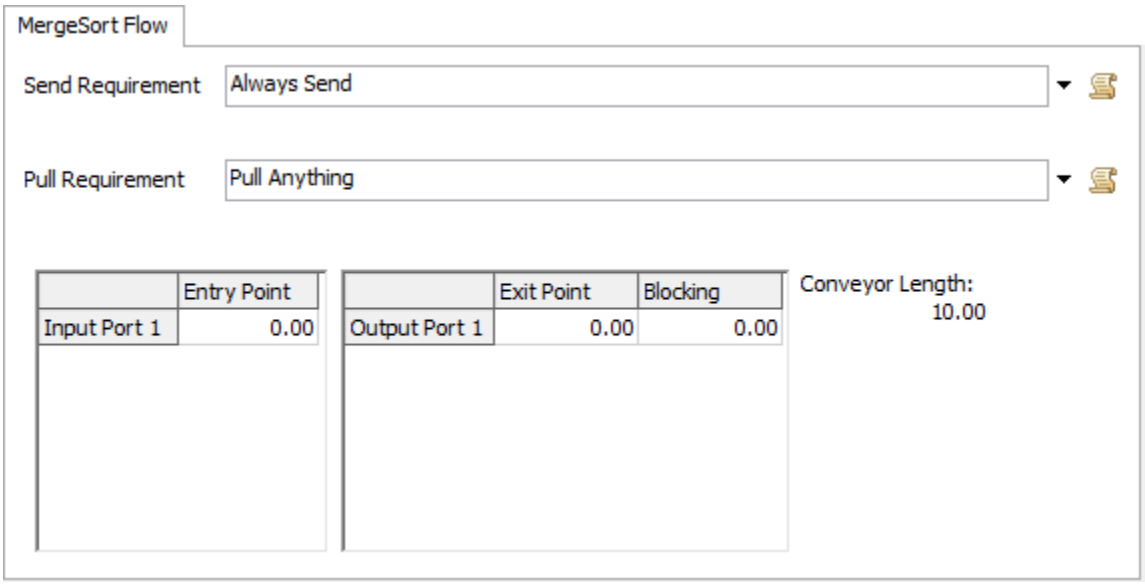
The table view contains most of the same functionality as the editor view, but also allows you to add the table to the MTEI for use with the Excel Interface.

Fields marked in red should not be edited. They can, however, give you feedback on the sections you edit. For example, the seclength column shows the total length of the section, given the parameters that you have specified in the other fields. The startlength column shows the total length of the conveyor up to but not including that section. These fields will be updated when you reset the model.

This Page is Used By

Conveyor
MergeSort
BasicConveyor

MergeSort Flow Page



Send Requirement - This picklist replaces the Send To Port picklist in other FixedResources' Flow page. It is fired when a flowitem passes an output position on the conveyor. It should return a true or false value (1 or 0) of whether to send the flowitem out that output port.

Pull Requirement - This picklist is the same as the Pull Requirement on other FixedResources' Flow page. See Pull Requirement picklist.

Output Port / Input Port tables - These tables allow you to define the position and blocking logic for transfer points on the conveyor. Every input port of the MergeSort has an associated entry location along the length of the conveyor. Every output port has an associated exit location and a blocking parameter. If you connect the input/output ports of the conveyor while this window is open, you will need to reopen the window in order for your changes to be registered. In your Ortho/Perspective view, the positions of the entry/exit points will be drawn as red or green arrows. Input locations have arrows drawn pointing into the conveyor. Exit locations have arrows pointing out of the conveyor. To hide these arrows, set the object to not show ports, either from its Properties window, or from the Edit Selected Objects. If your entry/exit points don't appear to be at the right locations, reset the model and they should be re-configured to the correct position.

Note on entry/exit positions: if you've changed the entry/exit positions of the mergesort, you will need to reset the model in order for those positions to be placed correctly.

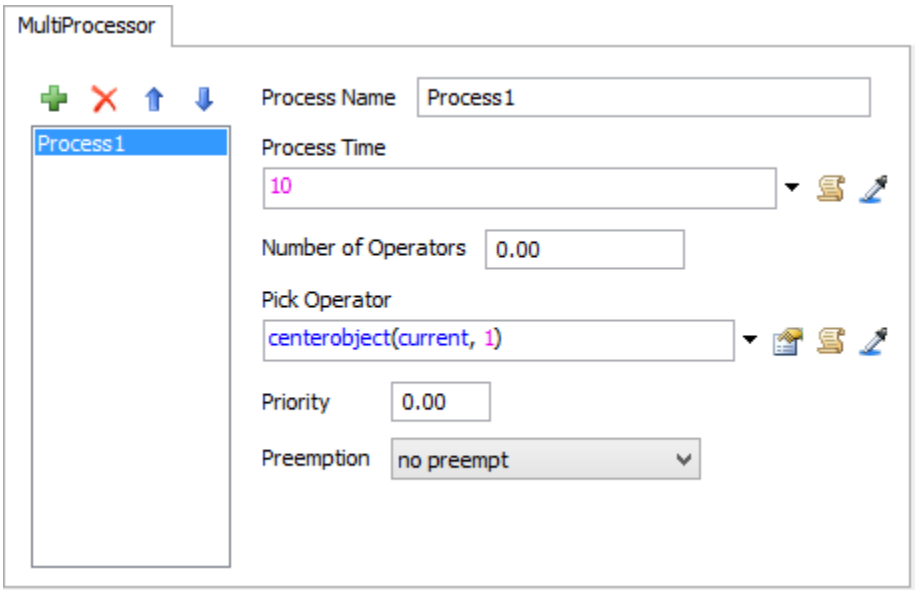
Entry/Exit Point - This is the position of the entry/exit point, measured from the start of the conveyor.

Blocking - This is found only in the Exit locations table. If this value is a 1, then whenever a flowitem passes that exit point and its Send Requirement returns 1, the conveyor will stop until that flowitem has exited. If this value is 0, the conveyor will "attempt" to send the flowitem out that exit point, but if the downstream object isn't ready to receive it, the flowitem will simply continue to the next exit point. We advise that the last exit point on the conveyor be blocking, or else flowitems that reach the end of the conveyor without exiting will simply start at the beginning of the conveyor again.

This Page is Used By

MergeSort

MultiProcessor Page



- Click this button to add a new process to the process list below.

- Click this button to delete a process from the process list below.

- Click this button to move the selected process up in the process list below.

- Click this button to move the selected process down in the process list below.

The process list contains each of the processes for this multiprocessor. To edit a process, highlight it in the list and change its name and other details to the right.

Process Details

For each process there will be a process tab or page to define the process. Process tabs will have the following information to define the process:

Process Name - The process name field allows the modeler to define the name of each process. This name will be used in the state reporting of the MultiProcessor.

Process Time - This picklist determines how long a processor spends processing a single flowitem. See also Cycle Time picklist.

Number of Operators - This number determines how many operators the object will use during its process time.

Pick operator - This picklist returns a reference to the operator or dispatcher that the object is using during the given process. See also Pick Operator picklist.

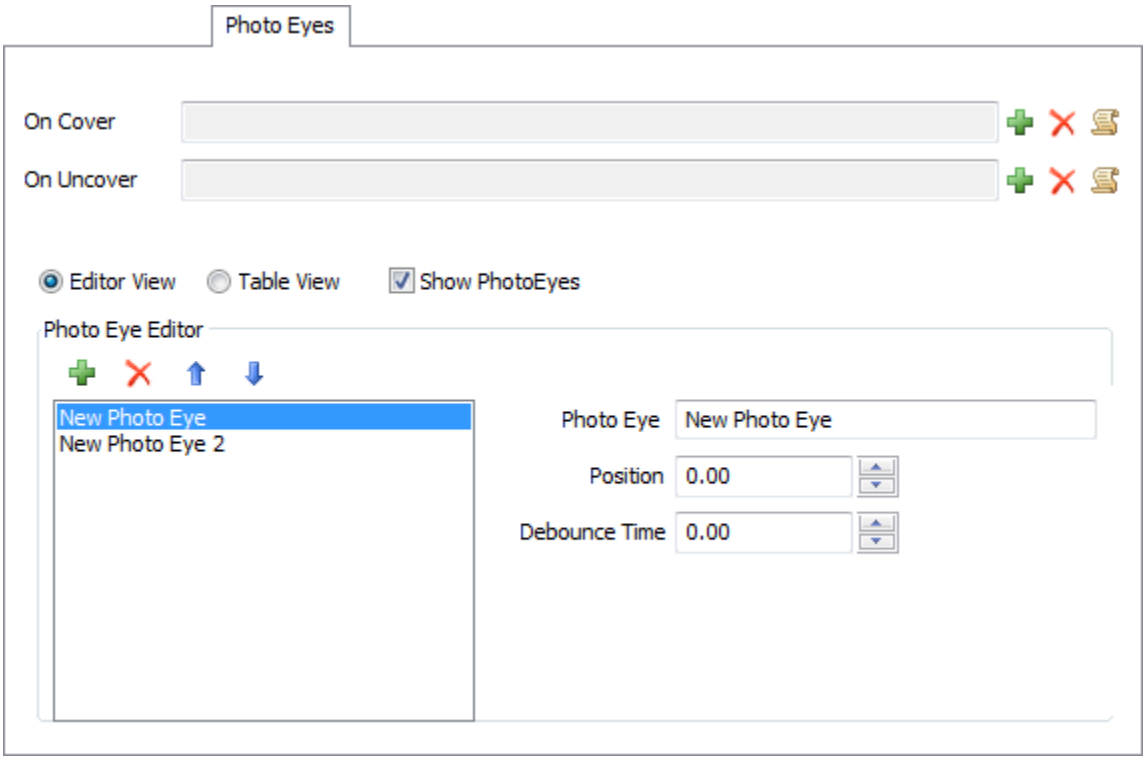
Priority - This value sets the priority of the task sequence that will be sent to the operator. Operators generally sort task sequences so that sequences with higher priorities will be performed first. Task sequences with the same priority will be performed in the order that they were received.

Preemption - Sets the preempt value for calling operators. See the Task Sequence Preempting page for more information on preemption.

This Page is Used By

MultiProcessor

Photo Eyes Page



For more detailed information, refer to the conveyor photo eye logic section.

OnCover - This picklist is fired in when a flowitem covers the photo eye and when the debounce time has expired without the photo eye being uncovered. See OnCover OnUncover Trigger.

OnUncover - This picklist is fired when a flowitem passes the photo eye with no flowitem behind it, uncovering the photo eye. See OnCover OnUncover Trigger.

Editor View - Displays the Photo Eye edit view as displayed above.

Table View - Displays the Photo Eye table view as displayed below.

Show PhotoEyes - Check this box if you want the conveyor's photo eyes to be shown in the 3D view.

- Adds a new Photo Eye to the Conveyor.

- Removes the selected Photo Eye from the Conveyor.

- Moves the selected Photo Eye up in the list. This does not affect the physical position of the Photo Eye along the conveyor, only its order in the list.

- Moves the selected Photo Eye down in the list. This does not affect the physical position of the Photo Eye along the conveyor, only its order in the list.

Photo Eye - This is the name of the Photo Eye. This is purely for the modeler's convenience and has no affect on the model.

Position - The position, in model units, along the conveyor that the Photo Eye is placed.

Debounce Time - Sets the debounce time in model time units. See the photo eye logic section for more information.

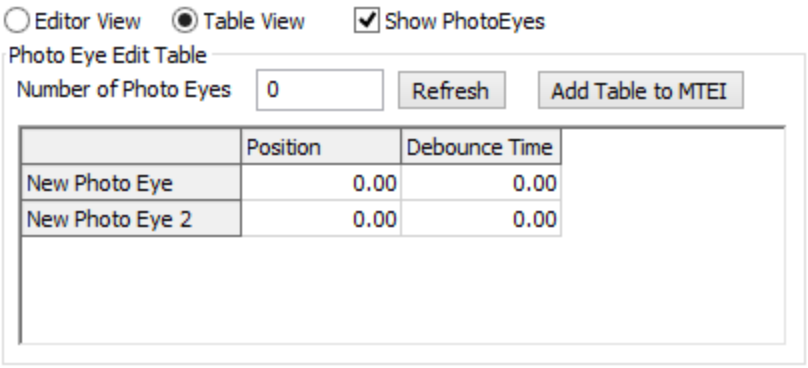


Photo Eye Edit Table - This table specifies the location and debounce time of each photo eye. The location is measured from the start of the conveyor.

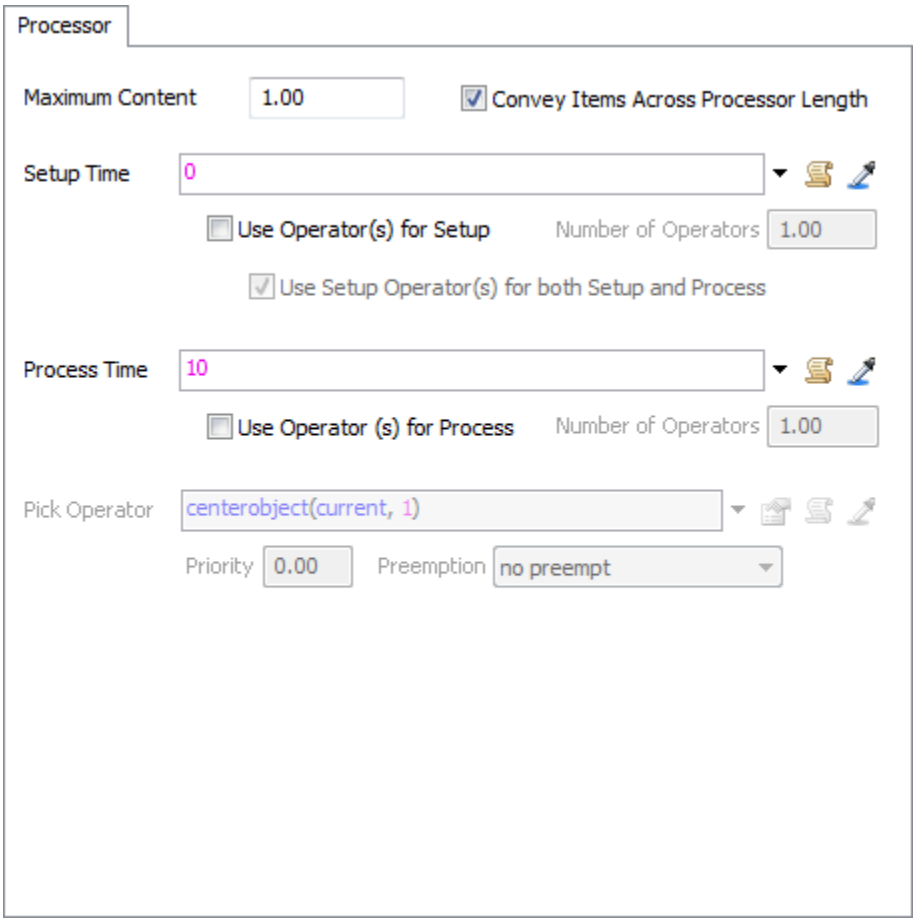
Number of Photo Eyes - This field specifies how many photo eyes to place along the length of the conveyor. Enter a number and press the Refresh button. The specified number of rows will appear in the Photo Eye table.

Add Table to MTEI - This buttons adds the table as a row in the Multiple Table Excel Import.

This Page is Used By

Conveyor

Processor Page



Maximum Content - This number defines the number of flowitems that the processor can hold at one time.

Convey Items Across Processor Length - If this box is checked, flowitems will be seen translating from one side of the processor to the other as their process time elapses. It is for visualization purposes only. If unchecked, entering flowitems will be placed in the middle of the processor and remain until exiting.

Setup

Setup Time - This picklist defines the amount of time that the processor waits after receiving a flowitem to begin processing that flowitem. See Setup Time pick list.

Use Operator(s) for setup - If this box is checked the object will call for one or more operators during its setup time. The operator(s) will be released after the setup time has expired.

Number of Operators - This parameter is only visible when the "use operator(s) for setup" box is checked. This number determines how many operators the object will use during its setup time.

Use the Setup Operator(s) for both Setup and Process - This parameter is only visible if both of the "use operator(s)" boxes are checked. If this box is checked, the operators that were called for setup time will be utilized during process time. If this box is not checked, the operators used for the setup time will be released and new operators will be called for the process time. Different operators can be called using a special pick option in the "Pick Operator" parameter.

Process

Process Time - This picklist determines how long a processor spends processing a single flowitem.

Use Operator(s) for process - If this box is checked the object will call for one or more operators during its processing time. The operator(s) will be released after the process time has expired.

Number of Operators - This parameter is only visible when the "use operator(s) for process" box is checked, and the "use setup operators for both setup and process" box is not checked. This number determines how many operators the object will use during its process time.

Pick Operator

These fields are only visible when either of the “Use Operator(s)” boxes is checked.

Pick operator - This picklist returns a reference to the operator or dispatcher that the object is using during setup or process time. See Process Dispatcher picklist.

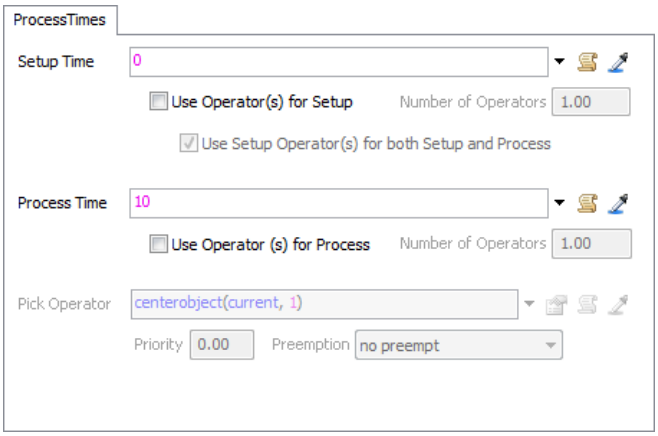
Priority -This value sets the priority of the task sequence that will be sent to the operator. Operators generally sort task sequences so that sequences with higher priorities will be performed first. Task sequences with the same priority will be performed in the order that they were received.

Preemption - Sets the preempt value for calling operators. This may cause the operator to perform tasks that would normally not be allowed. See the Task Sequence Preempting page for more information on preemption.

This Page is Used By

Processor

ProcessTimes Page



Setup

Setup Time - This picklist defines the amount of time that the processor waits after receiving a flowitem to begin processing that flowitem. See Setup Time pick list.

Use Operator(s) for setup - If this box is checked the object will call for one or more operators during its setup time. The operator(s) will be released after the setup time has expired.

Number of Operators - This parameter is only visible when the "use operator(s) for setup" box is checked. This number determines how many operators the object will use during its setup time.

Use the Setup Operator(s) for both Setup and Process - This parameter is only visible if both of the "use operator(s)" boxes are checked. If this box is checked, the operators that were called for setup time will be utilized during process time. If this box is not checked, the operators used for the setup time will be released and new operators will be called for the process time. Different operators can be called using a special pick option in the "Pick Operator" parameter.

Process

Process Time - This picklist determines how long a processor spends processing a single flowitem.

Use Operator(s) for process - If this box is checked the object will call for one or more operators during its processing time. The operator(s) will be released after the process time has expired.

Number of Operators - This parameter is only visible when the "use operator(s) for process" box is checked, and the "use setup operators for both setup and process" box is not checked. This number determines how many operators the object will use during its process time.

Pick Operator

These fields are only visible when either of the “Use Operator(s)” boxes is checked.

Pick operator - This picklist returns a reference to the operator or dispatcher that the object is using during setup or process time. See Process Dispatcher picklist.

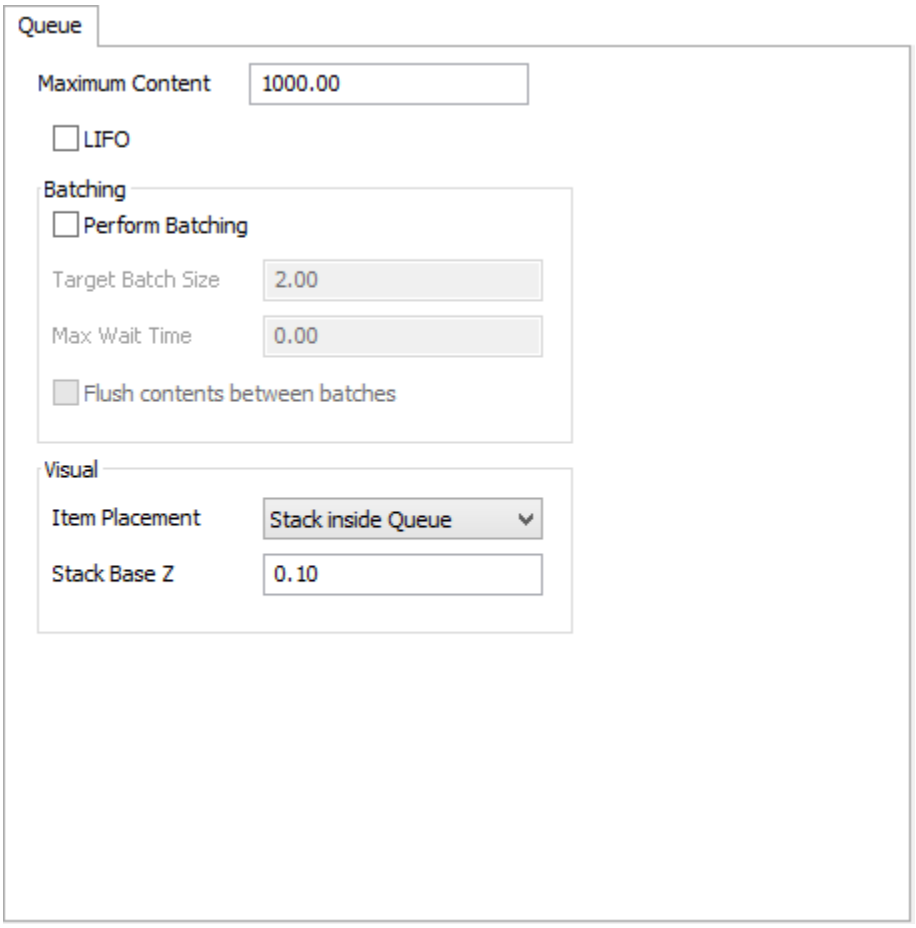
Priority - This value sets the priority of the task sequence that will be sent to the operator. Operators generally sort task sequences so that sequences with higher priorities will be performed first. Task sequences with the same priority will be performed in the order that they were received.

Preemption - Sets the preempt value for calling operators. This may cause the operator to perform tasks that would normally not be allowed. See the Task Sequence Preempting page for more information on preemption.

This Page is Used By

Combiner
Separator

Queue Page



Maximum Content - This is the maximum number of flowitems the queue can hold at once.

LIFO - If this box is checked the queue will act as a "last in first out" (LIFO) queue, otherwise it will act as a "first in first out" (FIFO) queue.

Batching

These fields define the queue's batching abilities.

Perform batching - If this box is checked, the queue will accumulate flowitems into a batch before releasing them downstream. Accumulation continues until either the target batch size is met or the max wait time expires. If this box is not checked, no batching will occur, and flowitems may leave as soon as downstream objects are available.

Target Batch Size - This number defines the size of the batches that the queue will gather before sending the flowitems downstream. Flowitems are sent downstream individually.

Max Wait Time - This number is the maximum length of time that the queue will wait before sending the flowitems downstream. If this time expires and the batch size has not been met, the currently collected batch will be released anyway. If 0 is specified in this field, then there is no maximum wait time, or in other words the queue will wait indefinitely.

Flush contents between batches - If this box is checked the queue will not allow new flowitems to enter until the entire current batch has left.

Visual

These properties define how the queue locates the flowitems within itself when they enter.

Item Placement - This defines how the flowitems are placed in the queue visually.

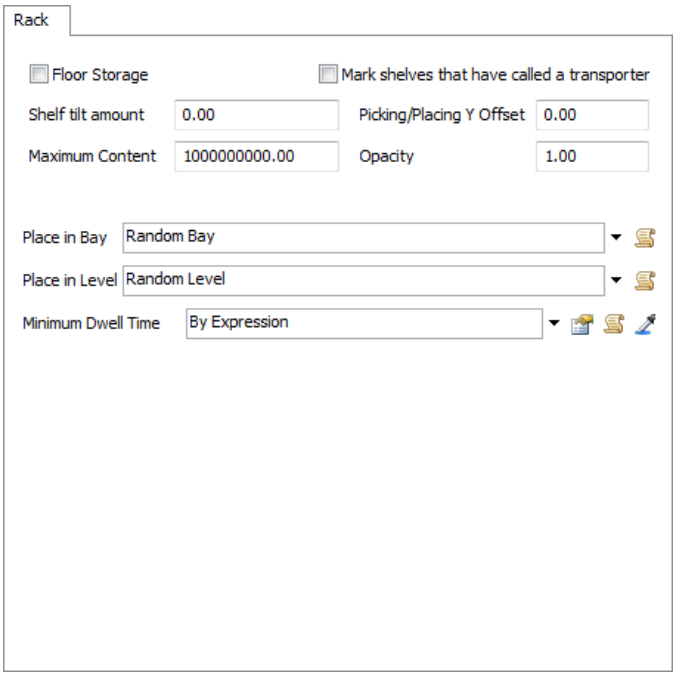
- **Stack Vertically -** The flowitems are stacked on top of each other. The flowitem at the bottom of the pile is the one that has been in the queue the longest.
- **Horizontal Line -** The flowitems are lined up horizontally. The one closest to the output ports of the queue is the one that has been in the queue the longest.
- **Stack inside Queue -** The flowitems are stacked in rows inside the queue. The flowitems' positions will move if a product ranked before them is taken out of the queue. If you would like the product positions to stay the same once they are in the queue, then have the queue be LIFO by having downstream objects pull only the last product in the queue.
- **Do Nothing -** Flowitems are all placed at the same point in the queue. This may make it appear as if the queue is only holding one flowitem.

Stack Base Z - This number defines the height where the queue begins placing flowitems that are being stacked vertically or inside the queue.

This Page is Used By

Queue

Rack Page



Floor Storage - If this box is checked, then instead of having a vertical storage rack, the rack will simulate storage space on the floor. Looking down from above the rack, bays are vertical columns, and levels are horizontal rows on the rack.

Mark Shelves that have called a transporter - This check will highlight the shelve in a red color when it has called the transporter for pickup.

Shelf Tilt Amount - This number defines the amount of tilt of items placed in a given cell of the rack, as some racks have products slide down from the back of the rack to the front.

Picking/Placing Y Offset - This value is used to configure how close transport objects come to the rack when they drop off or pick up flowitems from the rack. This is especially useful if operators are used to drop off and pick up from the rack, because often they will walk into the middle of the rack to get a flowitem. Specify a value of 1, for example, and the operators will keep better distance from the rack when dropping off and picking up flowitems.

Maximum Content - This number defines how many flowitems the rack will be allowed to hold at a given time.

Opacity - This value allows the drawing of the rack to be translucent, so that if there are several racks in the same area, many of them can be seen because of the translucency of the racks in front. A value of 0 means totally transparent, and a value of 1 means totally opaque.

Place in bay - This picklist is called when a flowitem is entering the rack. It returns which bay the flowitem will be placed in. See Place in Bay picklist.

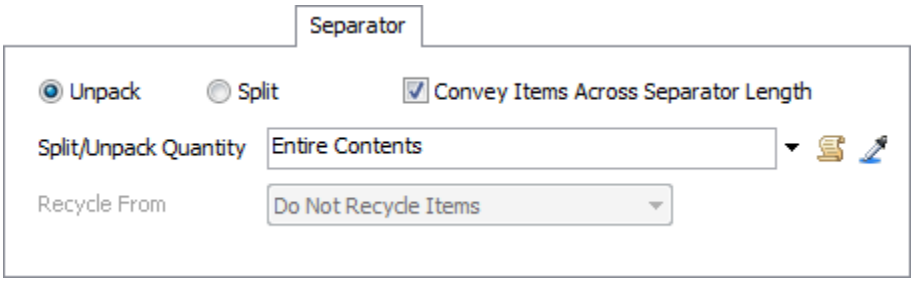
Place in level - This picklist is called when a flowitem is entering the rack. It returns which level the flowitem will be placed in. See Place in Level picklist.

Minimum Dwell Time - This picklist returns a value of how long a flowitem must stay in the rack before it is released to continue downstream. You can also return a value of -1 from this function so the Rack will not release the item at all, and then implement your own releasing strategy using the releaseitem() command. See Minimum Staytime picklist.

This Page is Used By

Rack

Separator Page



Unpack - If this button is checked, the separator will assume that the flowitem that entered contains other flowitems that need to be removed.

Split - If this button is checked, the separator will make duplicate copies of the entering flowitem.

Convey Items Across Separator Length - If checked, flowitems will travel across the Separator during their process time.

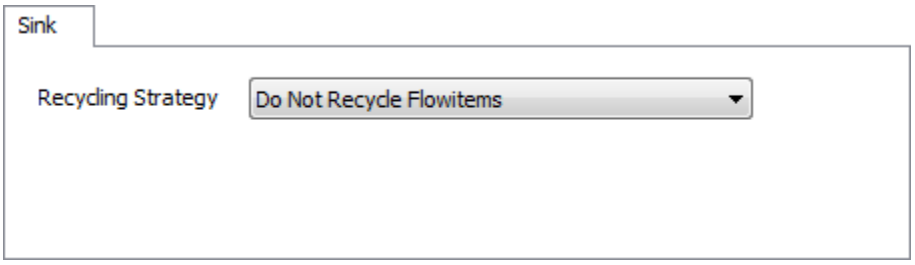
Split/Unpack Quantity - This picklist returns how many flowitems will be unpacked or duplicated by the separator. Refer to the split quantity picklist.

Recycle From (Split mode only) - In split mode, the flowitems that are created can be pulled from recycled flowitems rather than creating a new copy. You can use this option to specify what type of flowitem should be created. To learn more about recycling, refer to the Sink Page.

This Page is Used By

Separator

Sink Page



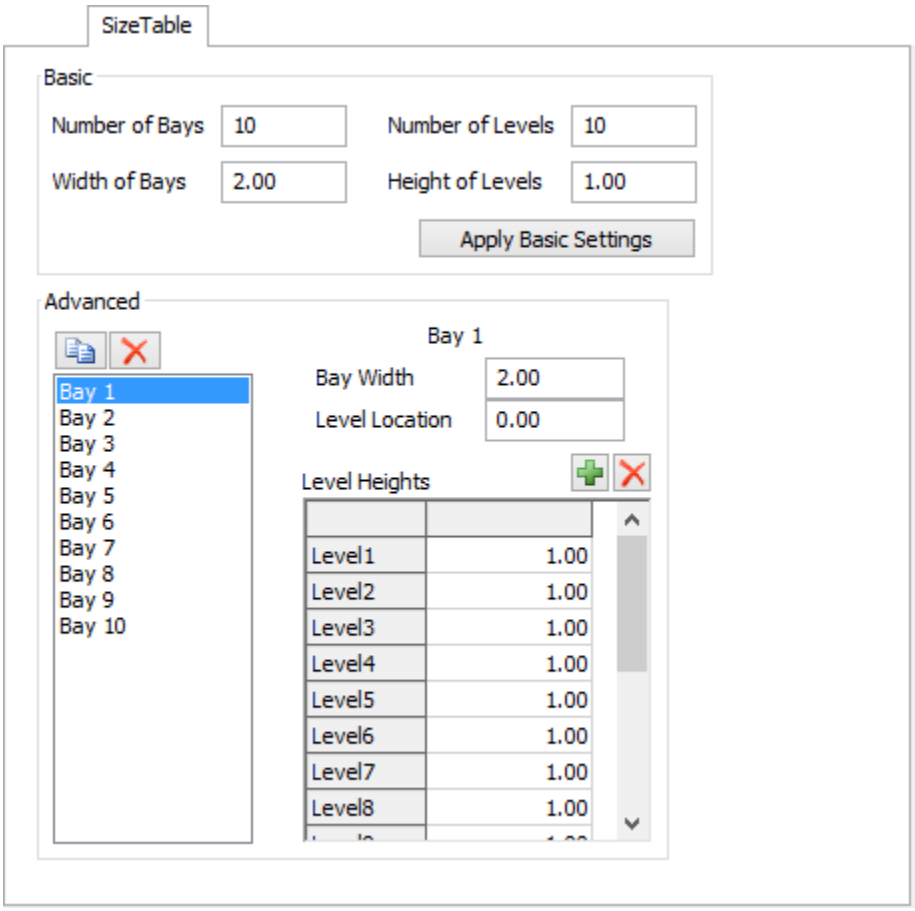
Recycling Strategy - The recycling strategy drop-down list lets you specify how the Sink recycles flowitems. Recycling flowitems can significantly increase the speed of your model. By default, the Sink simply destroys flowitems that enter. To configure the sink to recycle flowitems, select from the drop-down list a flowitem in the flowitem bin that this sink's flowitems originate from.

Note on recycling flowitems: If you configure a Sink to recycle flowitems, those flowitems will be returned to the flowitem bin as-is. This means that if there were changes made to the flowitems during the model run, you will need to change them back to their original state from the entry trigger of the sink, so that when they are recycled, they will have the correct data, visuals, etc.

This Page is Used By

Sink

SizeTable Page



This page allows you to configure the layout of bays and levels on the rack. If you want a simple grid of bays and levels, then you can specify these settings from the Basic panel at the top. If you want to specified different sizes for different bays and levels of the rack, then you can use the Advanced panel at the bottom to configure each bay individually.

Basic Panel

Use the basic panel if your rack is just a simple grid of uniform cells. You can also use it to set basic settings for the rack before going in and editing individual bays in the advanced panel. Once you have specified the basic dimensions for the rack, click the Apply Basic Settings button, and these settings will be applied to the rack.

Number of Bays - This value is for the number of bays (columns) for the rack.

Width of Bays - This value defines the default width of each bay.

Number of Levels - This value is for the number of levels in the rack.

Height of Levels - This value is for the default height of each level. This value can be edited for each level and bay using the advanced edit.

Advanced Panel

Use this panel to configure individual bays and levels on the Rack. On the left side of this panel is a list of all the bays in the rack. Select a bay and configure it by using the options and buttons on the right side of the panel. Changes should immediately be shown on the Rack in the orthographic/perspective view. If they do not show up immediately, hit the Apply button, and they should appear.

- This button duplicates the currently selected bay and adds it to the end of the rack.

- This button deletes the currently selected bay in the rack.

Bay Width - This field specifies the width of the currently selected bay.

Level Location - This field specifies the initial z location of the first level of the selected bay.

Level Heights Table - Here you can specify the height of each level in the currently selected bay.

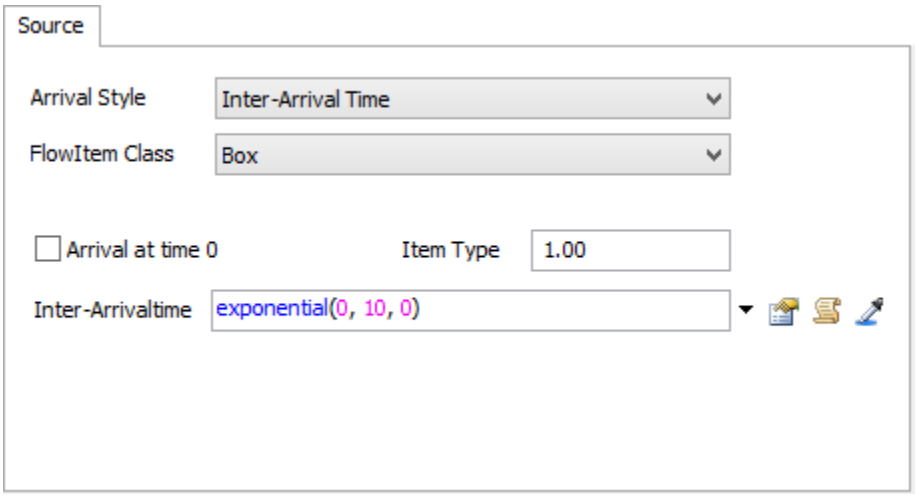
- This button adds a level to the end of the currently selected bay.

- This button deletes a level from the end of the currently selected bay.

This Page is Used By

Rack

Source Page



Arrival Style

This is used to specify the way that the source creates flowitems.

Inter-Arrival time - After a set period of time, the source creates one flowitem. This repeats until the model is stopped.

Arrival Schedule - The source follows a table that defines when flowitems are created, how many there are, and what itemtypes are assigned to them.

Arrival Sequence - The source follows a table that defines what order flowitems are created in. Flowitems are created as fast as the source can move them downstream.

FlowItem Class

This is used to define what class of flowitem the source will create. To view and edit flowitem classes, select *Go to Flowitem Bin...* in the drop down, or go to the Tools menu > Flowitem Bin.

Inter-Arrivaltime Usage

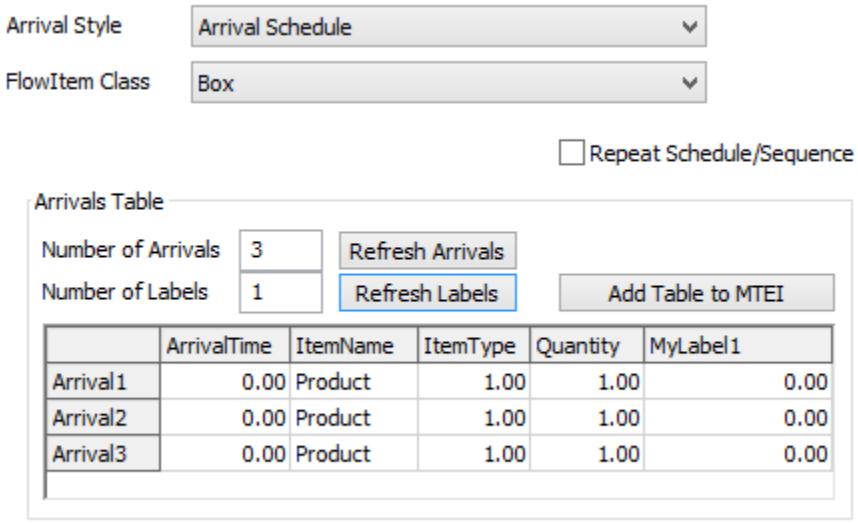
These fields define how the source creates flowitems when inter-arrival time is selected as the arrival style.

Arrival at time 0 - If this is checked, one flow item will be created at time 0. The next will be created at the end of the first inter-arrival time.

Item Type - Flow items that are created will be assigned the itemtype defined here.

Inter-Arrival time - A function that returns the amount of time the source should wait before creating the next flowitem.

Arrival Schedule/Sequence



These fields define how the source creates flowitems when Arrival Schedule or Arrival Sequence is selected as the Arrival Style.

Repeat Schedule/Sequence - If this box is checked, the schedule or sequence will continually repeat itself until the model is stopped. In the case of a schedule, the arrival defined in row 1 will occur 0 seconds after the time defined for the arrival in the last row. This means that the arrival time for the very first row of the table is only used once for a given simulation. Note that this initial delay can be used as a warm-up period. If you would like to specify an interval time between the arrival time of the last row and the arrival time of the first for when repeated, then add another row to the end of the table and specify a quantity of 0.

Number of Arrivals - This specifies how many rows are in the arrival table. Change the value in this field and hit Refresh Arrivals to update the table view.

Number of Labels - This specifies how many columns are in the arrival table. Change the value in this field and hit Refresh Labels to update the table view. Additional columns will be added for labels and their initial values that will be added to the flowitems when they are created.

Add Table to MTEI - This buttons adds the table as a row in the Multiple Table Excel Import.

ArrivalTime - (Arrival Schedule only) This is the time in model time units that the arrival will occur.

Item Name - Specifies the name of the flowitem when it's created.

ItemType - The itemtype the flowitems will be given when they're created.

Quantity - This number specifies how many flowitems will be created during this arrival.

Labels - This specifies all labels that will be added to created flowitems and their initial values.

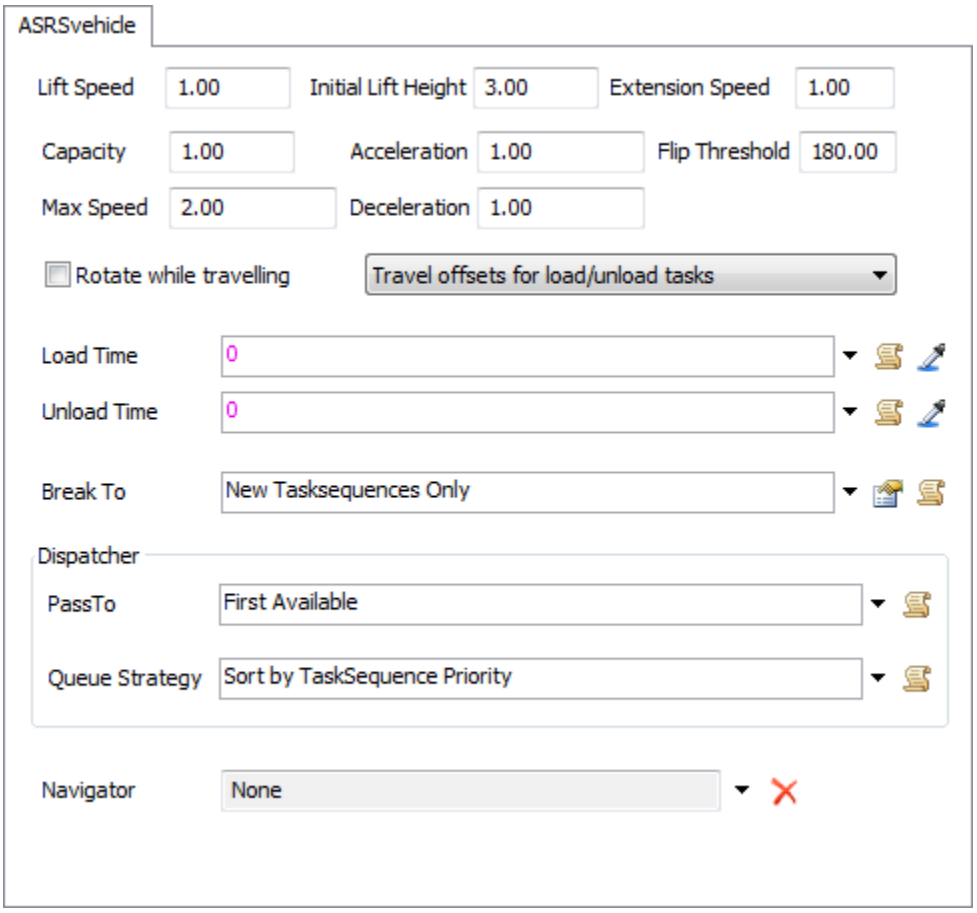
This Page is Used By

Source

TaskExecuter Properties Pages

1. **ASRSvehicle**
2. **BasicTE**
3. **Breaks**
4. **Collision**
5. **Crane**
6. **Dispatcher**
7. **Geometry**
8. **Robot**
9. **TaskExecuter**
10. **Transporter**

ASRSvehicle Page



Lift speed - This number is how fast the lift on the ASRSvehicle moves up and down.

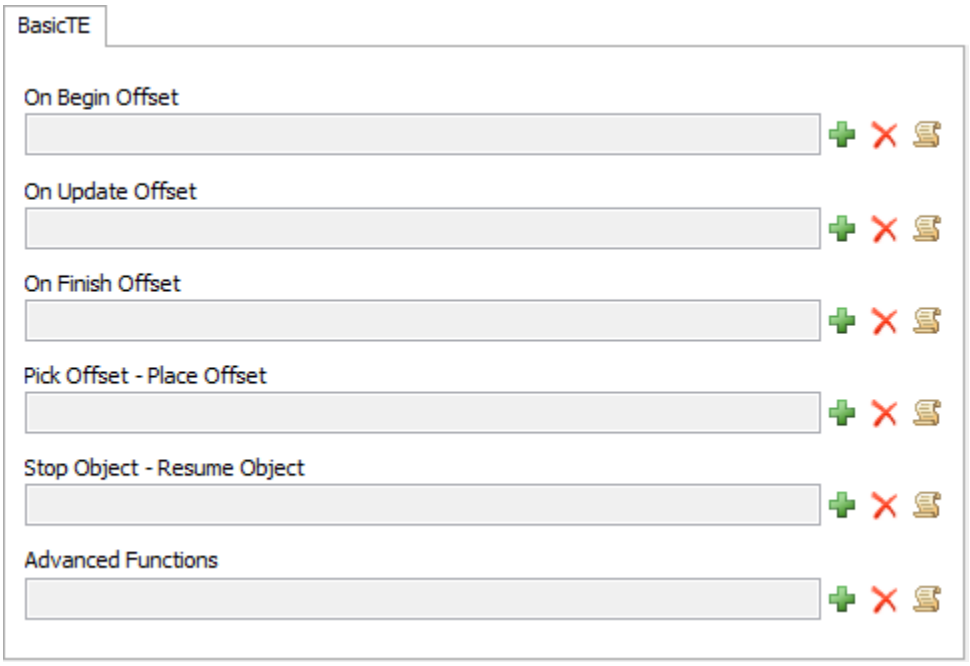
Initial lift height - This number defines the reset z location of the ASRSvehicle's platform. The platform returns to this height when the model resets. It also returns to this height when it is not traveling the offset of a load/unload task.

Task Executer fields - This page has many of the same controls as the TaskExecuter Page. They are described in more detail there.

This Page is Used By

ASRSvehicle

BasicTE Page



Note: For more information on offset travel, refer to the TaskExecuter's documentation on offset travel.

OnBeginOffset - This picklist is fired at the beginning of an offset travel operation. An x,y,z offset location is passed into this function. From this x,y,z offset location, the object should figure out how it will travel the specified offset, and then return the amount of time it will take to make the travel operation.

Access variables for the OnBeginOffset function are as follows:

current: the current object

x: the requested x offset. This is an offset distance from the x center of the object.

y: the requested y offset. This is an offset distance from the y center of the object.

z: the requested z offset. This is an offset distance from the z base of the object.

item: if the offset operation has an involved item, this is a reference to the item

endspeed: this is the requested end speed for the offset travel operation

maxspeed: this is the value of this object's maximum speed variable

acceleration: this is the value of this object's acceleration variable

deceleration: this is the value of this object's deceleration variable

lastupdatedspeed: this is the value of this object's lastupdatedspeed variable. It is the end speed of the object's last travel operation.

rotatewhiletraveling: this is the value of this object's rotatewhiletraveling variable. It is a 1 or 0, and specifies whether the user wants the object to rotate while traveling.

OnUpdateOffset - This picklist is fired before the view is refreshed. Here is where the object updates its location based on its current offset operation.

OnFinishOffset - This picklist is fired when the object has finished its offset operation. This should update the location of the object to its final offset location.

Access variables for the OnUpdateOffset function and the OnFinishOffset function are defined as follows:

current: the current object

offsettingnow: this is the value of this object offsettingnow variable. If the object is currently doing an offset operation, this will be 1, otherwise 0. The offsettingnow value is automatically set to 1 when OnBeginOffset is called, and then set back to 0 when OnFinishOffset is called.

offsettingtotaltime: this is the value of this object offsettotaltime variable. It tells the total time this object returned from its OnBeginOffset function.

maxspeed: this is the value of this object's maximum speed variable

acceleration: this is the value of this object's acceleration variable

deceleration: this is the value of this object's deceleration variable

lastupdatedspeed: this is the value of this object's lastupdatedspeed variable. It is the end speed of the object's last travel operation.

rotatewhiletraveling: this is the value of this object's rotatewhiletraveling variable. It is a 1 or 0, and specifies whether the user wants the object to rotate while traveling.

curloadunloadtime: if the offset operation is a load or unload operation, then this value is the load/unload time for the operation. Note that if there is a non-zero load/unload time, then this time will be executed before the OnFinishOffset trigger is fired. This means that in your update offset function, you may need to query whether you're in the offset part of the operation, or in the load/unload part of the operation. Usually this will not matter, though, because the travel operations will finish, and during the remaining load/unload time, the update function will automatically set the object's location to the final destination location. You will really only need to use this if you are going to do some animation/movement during the load/unload time, as the ASRSvehicle and Elevator objects do.

Pick Offset - Place Offset-

This picklist is fired at two different times. First is when another transport object is attempting to place, or unload, a flowitem into this object. This is called place offset. It should return an offset location for the transport to offset to before placing the item. The second time this is called is when a transport is about to pick, or load, an item from this object. This is called pick offset. It should again return an offset location for the transport to offset to before picking the product up. Note that this function does not fire when this object is attempting to load/unload to or from another object, but rather when another object is attempting to load/unload to or from this object.

Access variables for the pick offset - place offset function are as follows:

current: the current object

pickoperation: this variable is 1 or 0. 1 means it is a pick operation, 0 means it is a place operation.

item: this is a reference to the item that is being picked or placed

otherobject: this is a reference to the object that is picking or placing the item

xvalnode, yvalnode, zvalnode: these parameters are references to nodes whose values should be set in this function, and will represent the offset location returned by this function. For example, if I want the picking/placing object to offset 10 in the x direction, 0 in the y direction, and 5 in the z direction, I would set xvalnode to 10 using the setnodenum() command, yvalnode to 0, and zvalnode to 5.

Note on the pick/place offset return value: If you are implementing your own pick/place logic here, and do not want the default logic to be executed, you must return 1 from this function. If the object gets a return value of 0 from this function, it will assume that nothing was done, and will execute its own default logic.

Stop Object - Resume Object -

This picklist is fired at two different times. First is when the stopobject() command is called for this object. Second is when the resumeobject command is called for this object. This field should define a strategy for how the object will "stop", and how it will "resume". This field should also manage data for remembering how many stop requests have been made, what the state of the object was before it was stopped, etc.

Access variables for the stop object - resume object function are as follows:

current: the current object

stopoperation: this value is either a 1 or 0. 1 means the stopobject() command is being called on this object. 0 means the resumeobject() command is being called on this object.

stopstate: this value is only applicable for a stop operation. It specifies the requested state passed into the stopobject() command.

nrofstopsnode: this is a reference to this object's nrofstops variable. If this is a stop operation, the value of this node should be incremented by 1 within this function. If it is a resume operation, the value of this node should be decremented by 1 within this function. Also, you will get the value of this node to know how many stopobject() commands have been called on this object. When the object is stopped for the first time (the value of the node goes from 0 to 1), you should execute logic specifying how to stop this object. When the object is resumed for the final time (the value of the node goes from 1 to 0), you should execute logic specifying how to resume this object.

timeoflaststopnode: this is a reference to this object's timeoflaststop variable. When this object is stopped for the first time, this node should be set to the current time, so you can know the amount of time the object was stopped when it is finally resumed.

statebeforestopnode: this is a reference to this object's statebeforestopped variable. When this object is stopped for the first time, this node should be set to the object's current state, so that you can know which state to go back to when the object is resumed.

Note on the stop/resume object return value: If you are implementing your own stop/resume logic here, and do not want the default logic to be executed, you must return 1 from this function. If the object gets a return value of 0 from this function, it will assume that nothing was done, and will execute its own default logic.

Advanced Functions - This picklist is fired for several different notifications or functions that are called on the object. For the most part, you will not need to implement any logic for these notifications, but they are nonetheless made accessible to you. The return value of this function should either be a 1 or a 0. If 0, the object will execute the default functionality associated with the given notification. If 1, the object not do any default functionality, but assumes that the function has overridden the default.

The type of notification called for the advanced function is passed in as parval(1), or msgtype. This parameter can be one of several values. These values are listed as follows:

ADV_FUNC_CLASSTYPE: This is a request to get the type of class of the object. The classtype should be returned as an integer with certain bits set high. You can construct this value using the bitwise OR operator | and several classtype macros. For example, a FixedResource's classtype is: CLASSTYPE_FLEXSIMOBJECT > CLASSTYPE_FIXEDRESOURCE.

ADV_FUNC_DRAGCONNECTION: This function is called when a keyboard key is held down, and the user clicks and drags from this object to another object. In this case, the object to which the mouse was dragged is passed in as parnode(2), the ascii value of the key that was clicked is passed in as parval(3), and the classtype value of the object is passed in as parval(4).

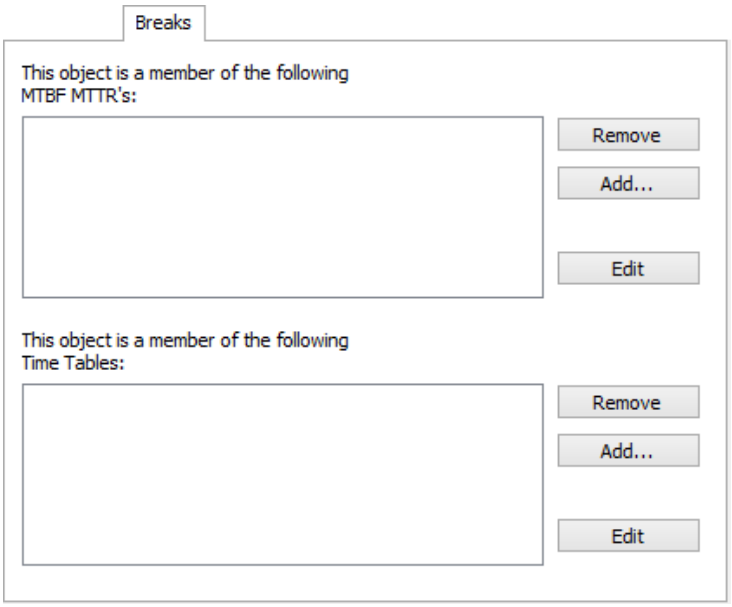
ADV_FUNC_CLICK: This function is called when the object is clicked on. Here a reference to the view in which it was clicked is passed in as parnode(2), and the click code is passed in as parval(3). Possible click codes are: DOUBLE_CLICK, LEFT_PRESS, LEFT_RELEASE, RIGHT_PRESS, RIGHT_RELEASE.

ADV_FUNC_KEYEDCLICK: This function is called when a key on the keyboard is held down and the object is clicked on. Here the view is passing as parnode(2), the click code is passed in as parval(3), and the ascii value of the pressed key is passed in as parval(4). Possible click codes are: DOUBLE_CLICK, LEFT_PRESS, LEFT_RELEASE, RIGHT_PRESS, RIGHT_RELEASE.

This Page is Used By

BasicTE

Breaks Page



MTBF MTTR Member List - This is a list of all the MTBF MTTR objects that have this object as one of its members. Each MTBF MTTR object can be connected to more than one object in the model. And each object can be controlled by more than one MTBF MTTR object. For more information about MTBF MTTR objects, refer to the Modeling Tools section about MTBF/MTTR objects.

Remove - This button removes the object from the selected MTBF MTTR object's member list.

Add.. - This button opens a listbox of all the MTBF MTTR objects in the model. You can select an MTBF MTTR object from the list to add this object to that MTBF MTTR object. You can also select "Add New MTBF MTTR" to create a new MTBF MTTR object in your model and add this object to that MTBF MTTR object.

Edit - This button allows you to edit the MTBF MTTR object's properties, including the timing of the breakdowns and repairs. For more information about MTBF MTTR objects, refer to the Modeling Tools section about MTBF/MTTR objects.

Time Tables Member List - This is a list of all the Time Table objects that have this object as one of its members. Each Time Table object can be connected to more than one object in the model. And each object can be controlled by more than one Time Table object. For more information about Time Table objects, refer to the Modeling Tools section about Time Tables.

Remove - This button removes the object from the selected Time Table object's member list.

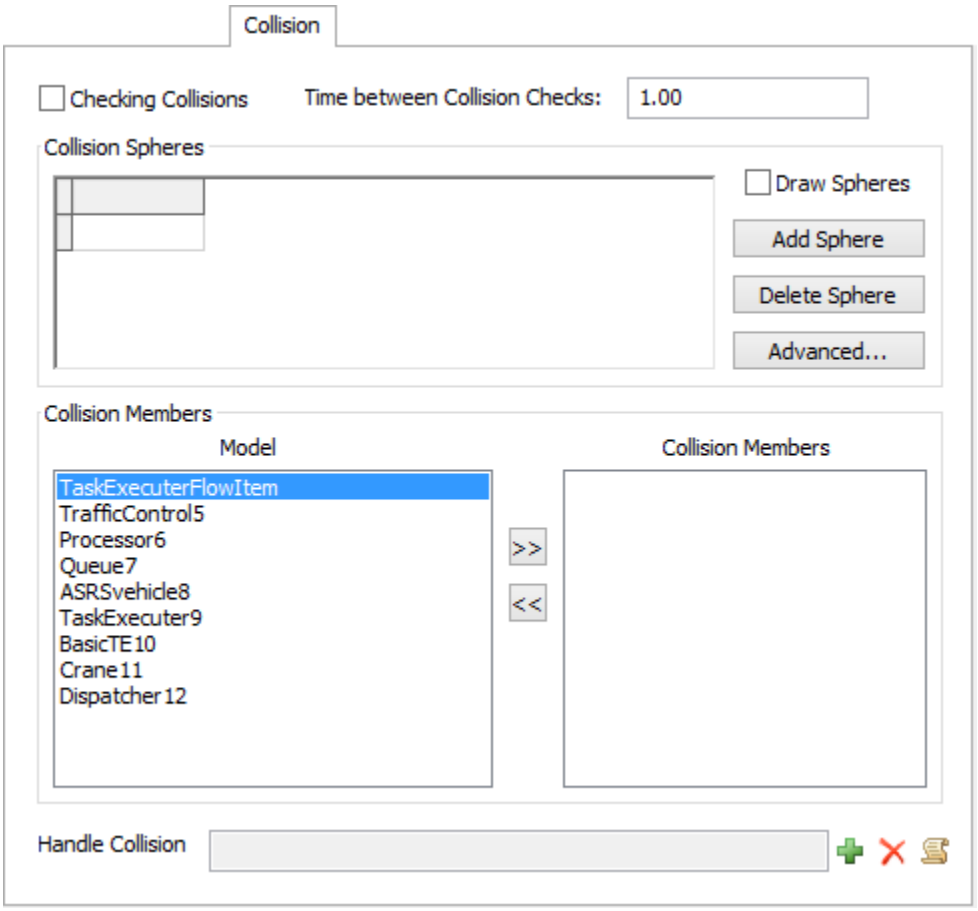
Add.. - This button opens a listbox of all the Time Table objects in the model. You can select an Time Table object from the list to add this object to that Time Table object. You can also select "Add New Time Table" to create a new Time Table object in your model and add this object to that Time Table object.

Edit - This button allows you to edit the Time Table object's properties, including the timing and duration of scheduled breakdowns. For more information about Time Table objects, refer to the Modeling Tools section about Time Tables.

This Page is Used By

TaskExecuter
Transporter
Operator
Crane
ASRSvehicle
Robot
Elevator

Collision Page



For more information on collision detection functionality, refer to the TaskExecuter.

Checking Collisions - Check this box to turn on collision detection. time between its collision members' collision checks.

Time between Collision Checks - The simulation time that passes between this object's collision checks. This does not specify the time between its collision members' collision checks.

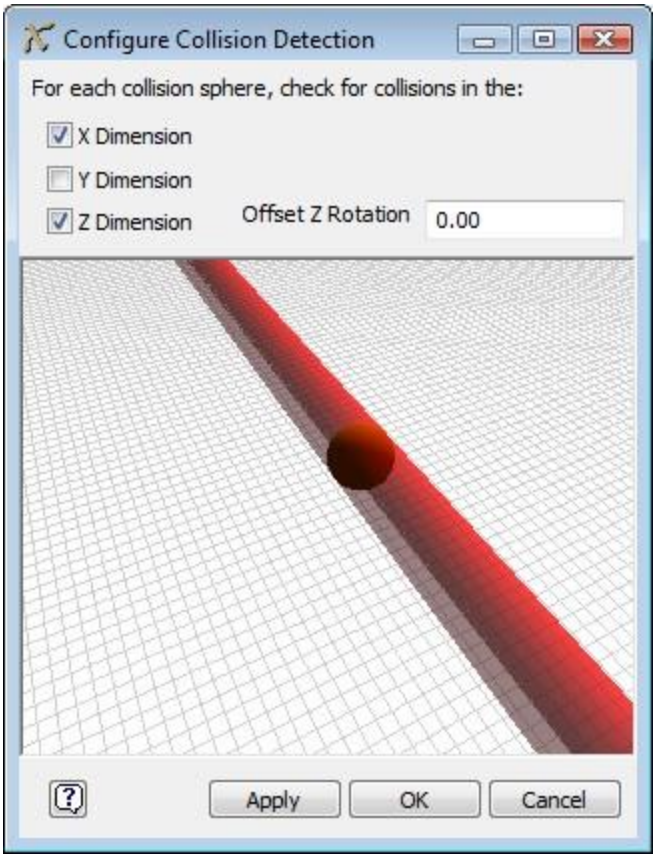
Collision Spheres - This table is used to define one or more collision spheres on the object.

Draw Spheres - Check this box if you want the collision spheres visible around the CollisionObject.

Add Sphere - Select this button to add a new sphere to the object. Define the size and position of the sphere in the table.

Delete Sphere - Select this button to delete the last sphere in the table.

Advanced... - This button allows you to optimize for collision checking speed by configuring the TaskExecuter to exclude certain axes when checking its spheres for collisions. This let's you cover more checking area with less spheres. Pressing the button opens the window below. Uncheck the X, Y, and/or Z Dimensions to exclude certain axes in the check. The result for your configuration is drawn in the view. A transparent cylinder or plane covers areas that will cause a collision given the configuration you've chosen. You can also enter an offset rotation, like 45 degrees, if you want to check for collisions on an axis that is not parallel with the normal axis. Note that the X, Y, and Z dimensions are according to the global coordinate system, and not according to the individual object's coordinate system.



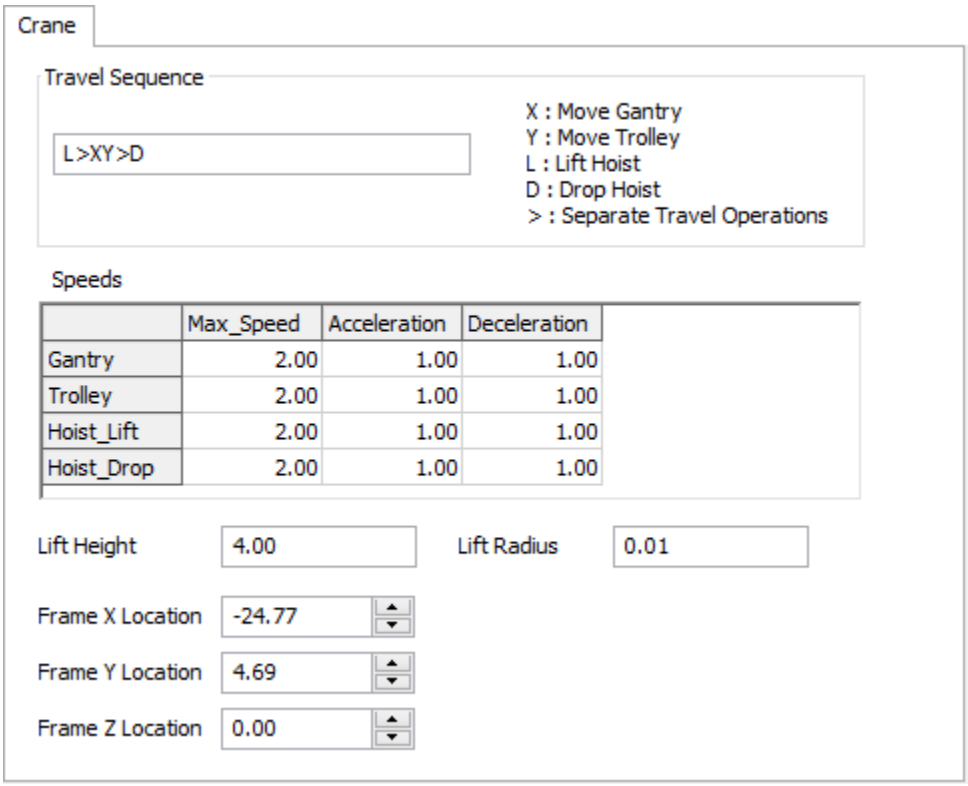
Collision Members - On the left is a list of model objects that can be added to the TaskExecuter's collision members. On the right is the list of collision members for the object. To add a member from the model list to the TaskExecuter's member list, select the object you want by clicking on it and then click on the >> button. To delete an object from the member list, select the one you want and click on the << button.

HandleCollisions - The picklist allows the user to define what happens when a collision takes place. See Collision Trigger picklist.

This Page is Used By

TaskExecuter
Transporter
Operator
Crane
ASRSvehicle
Robot
Elevator

Crane Page



Travel Sequence - Here you can specify the order in which the crane performs travel operations. Refer to the crane specifications for more information.

Speeds Table - Here you specify the max speed, acceleration and deceleration for each of the 4 operations the crane will do. Note that these operations only apply to offset travel. If the crane is connected to a network, then when it is on the network, only the normal maxspeed, acceleration and deceleration specified in the TaskExecuter page will be used.

Lift Height - Here you define how high the crane will lift to get to its lift height.

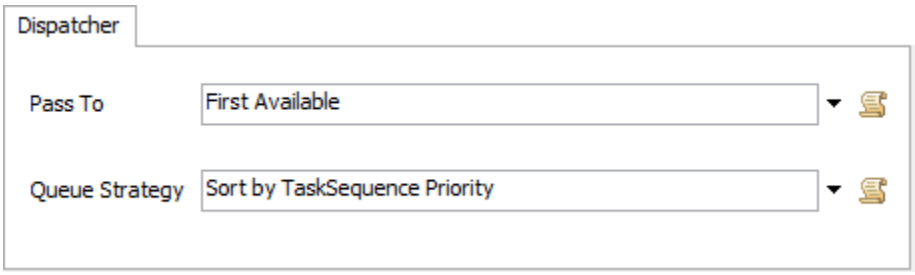
Lift Radius - Specify a radius within which the crane will not do a lift operation.

Frame X/Y/Z Location - These numbers define the location of the crane's frame. Note that this is different than the cranes actual x/y/z location. The crane's x/y/z location describes the location of the moving part of the crane. The frame will be stationary throughout the simulation, while the actual x/y/z location of the crane changes as the crane travels.

This Page is Used By

Crane

Dispatcher Page



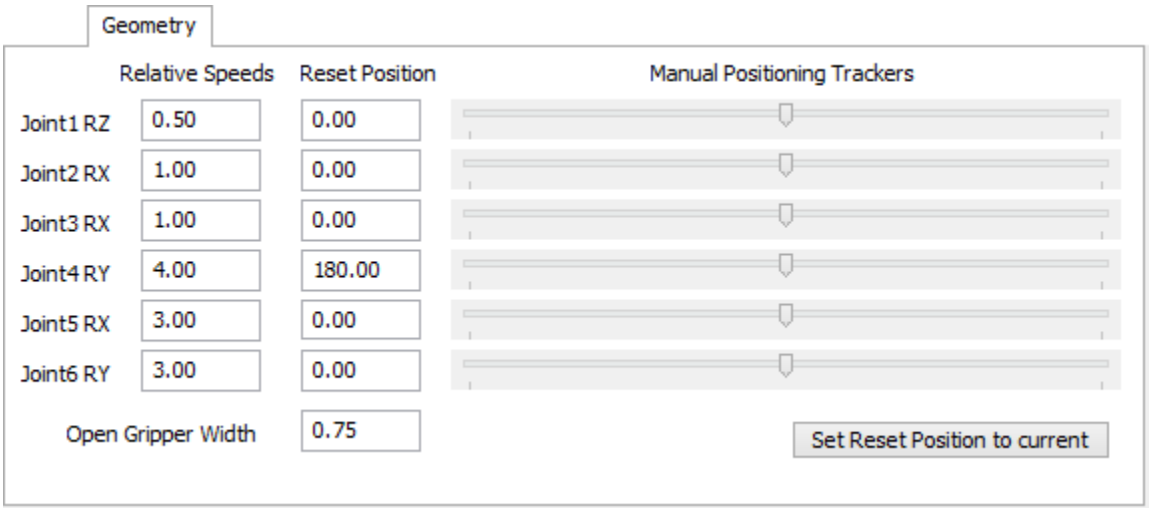
Pass To - This picklist returns the output port number that the task sequence should be dispatched to. If 0 is returned, then the task sequence will be queued up using the below mentioned queue strategy, and then will be dispatched to the first available mobile resource. If -1 is returned, then the Dispatcher will do absolutely nothing. In such a case you would use the movetasksequence() and dispatchtasksequence() commands to execute dispatching logic yourself. See Pass To picklist.

Queue Strategy - This picklist returns a "priority" value for the task sequence that is used to rank it in the object's task sequence queue. By default, it will simply return the priority value given to the task sequence when it was created, but the user can also customize task sequence priorities in this field. See Queue Strategy picklist.

This Page is Used By

Dispatcher

Geometry Page



Relative Speeds- The options in this column control the relative speeds for each joint..

Reset Position- The options in the column control the position to which the robot will return on reset.

Manual Positioning Trackers- The sliders in this column control the current position of each joint.

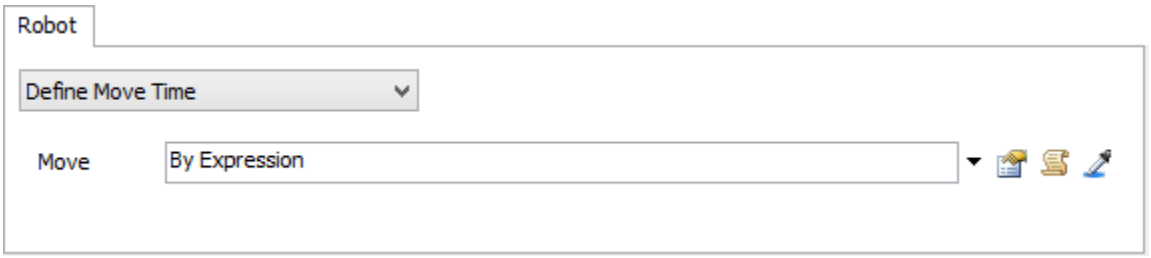
Open Gripper Width- This option controls how far open the gripper is when it's open.

Set Reset Position to current- This option saves the current position of the robot as its reset position.

This Page is Used By

Robot

Robot Page

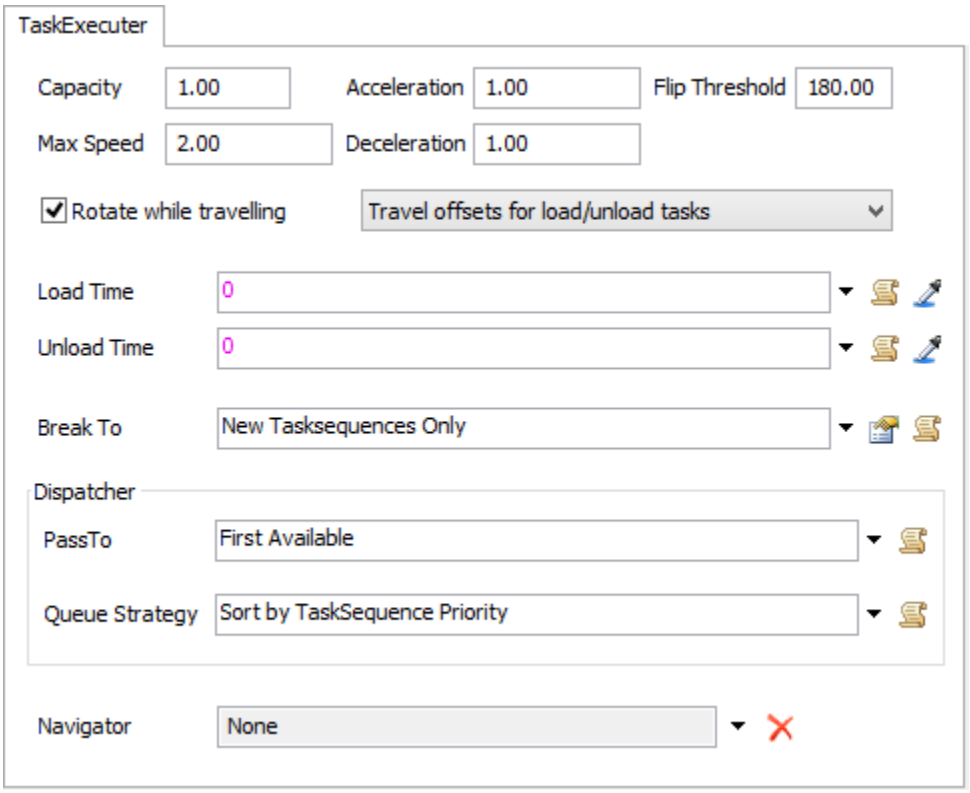


For more information, refer to the Robot topic.

This Page is Used By

Robot

TaskExecuter Page



Capacity - This number is maximum number of flowitems that the Task Executer can carry at a given time.

Max speed - This is the fastest that the Task Executer can travel.

Acceleration - This number is how fast the Task Executer gains speed until it reaches its maximum speed or needs to slow down to reach its destination node.

Deceleration - This number is how fast the Task Executer loses speed as it approaches its destination.

Flip Threshold - When the angle between the transporter/operator and the destination node meets or exceeds this value, the transporter/operator will flip (mirror image) in order to be facing the correct direction. This option will not affect the statistics of the model if checked or unchecked. It is simply for visualization.

Rotate while traveling - If this box is checked, the transporter/operator will rotate as needed in order to orient itself in the direction of travel. If the box is not checked, it will always face the same direction. This option will not affect the statistics of the model if checked or unchecked. It is simply for visualization.

Travel offsets for load/unload tasks - This box provides 3 options. If "Travel offsets for load/unload tasks" is selecteded, the transporter/operator will be move to the exact point where the flowitem is being picked up or dropped off. If "Do not travel offsets for load/unload tasks", it will travel to the origin of the destination object and pick up or drop off the flowitem there. In the case where the transporter/operator is using networknodes to travel to the destination, it will travel to the networknode attached to the destination object and then stop there. The option "Do not travel offsets and block space on network" only applies when the object is connected to a network of nonpassing paths. If it is chosen, then the object will arrive at the node, finish its travel, and while it is doing the load/unload operation, it will continue to take up space on the network, and block other objects traveling on the path.

Load time - This picklist returns how long it takes this Operator or Transporter to load the flowitem.

Unload time - This picklist returns how long it takes this Operator or Transport to unload the flowitem.

Break to Requirement - This field is executed when the TaskExecuter comes to a break task or callsubtasks task. The return value is a reference to a task sequence. The logic within this field should search the TaskExecuter's task sequence queue, and find a task sequence that is appropriate to break to.

Dispatcher Pass To - This picklist returns the output port number that the task sequence should be dispatched to. If 0 is returned, then the task sequence will be queued up using the below mentioned queue strategy, and then will be dispatched to the first available mobile resource. If -1 is returned, then the Dispatcher will do absolutely nothing. In such a case you would use the movetasksequence() and dispatchtasksequence() commands to execute dispatching logic yourself. See Pass To picklist.

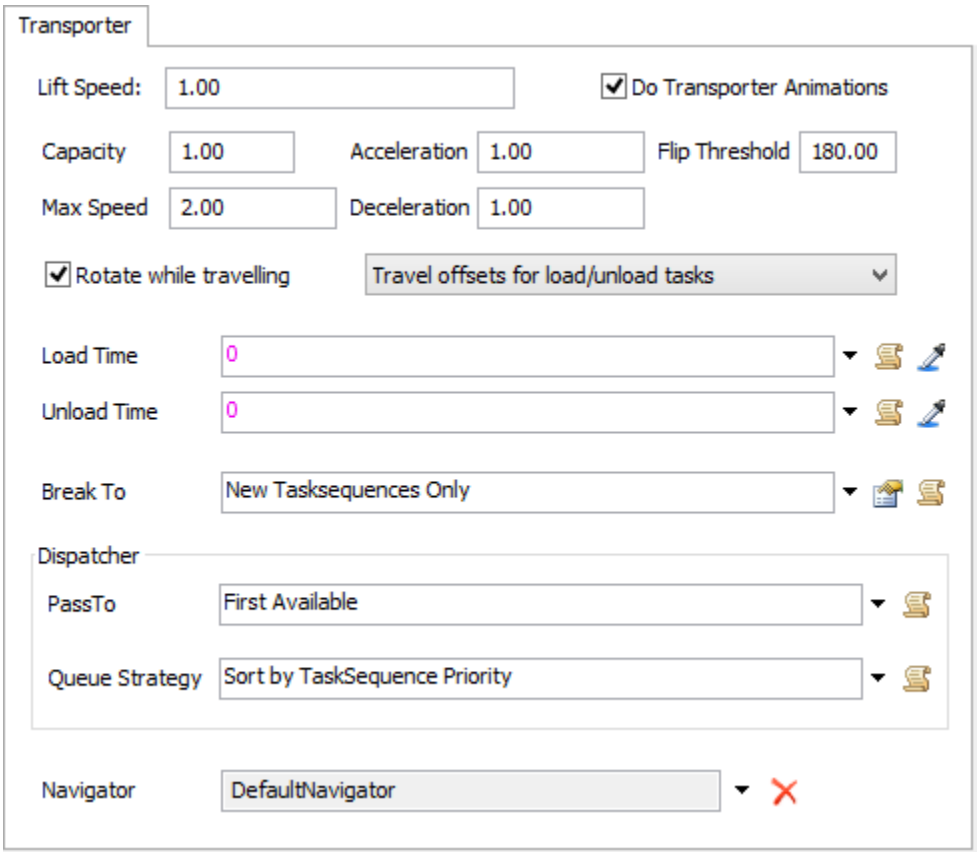
Dispatcher Queue Strategy - This picklist returns a "priority" value for the task sequence that is used to rank it in the object's task sequence queue. By default, it will simply return the priority value given to the task sequence when it was created, but the user can also customize task sequence priorities in this field. See Queue Strategy picklist.

Navigator - This specifies which Navigator the Task Executer object will use. If Navigator's are available in the model, they will be displayed in the drop down. To remove the TaskExecuter from using any Navigator, press the .

This Page is Used By

TaskExecuter
Transporter
Operator
Crane
ASRSvehicle
Robot
Elevator

Transporter Pages



Lift speed - This number is how fast the lifts on the Transporter move up and down.

Task Executer fields – This page has many of the same controls as the TaskExecuter Page. They are described in more detail there.

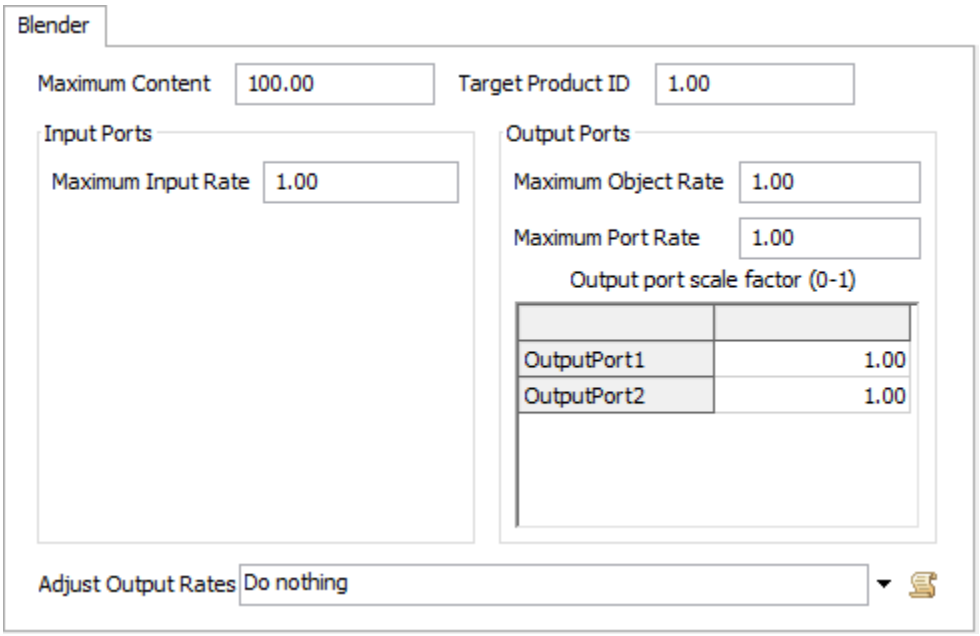
This Page is Used By

Transporter

Fluid Properties Pages

1. Blender
2. FluidConveyor
3. FluidLevelDisplay
4. FluidProcessor
5. FluidToItem
6. Generator
7. Initial Product
8. Inputs / Outputs
9. ItemToFluid
10. Marks
11. Mixer
12. Percents
13. Pipe
14. Pipe Layout
15. Recipe
16. Sensors
17. Splitter
18. Steps
19. Tank
20. Terminator
21. Ticker

Blender Page



Maximum Content - The maximum amount of fluid material that this object can hold at any time.

Target Product ID - The ProductID that will be assigned to the material that leaves this object.

Adjust Output Rates - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Input Ports

These properties affect how the object receives material from upstream objects.

Maximum Input Rate - The maximum rate that material will be allowed into this object through all input ports combined. The actual input rate is based on the amount of material available upstream and the space available in this object.

Output Ports

These properties affect how the object sends material to downstream objects.

Maximum Object Rate - The maximum rate that material will leave this object through all output ports combined.

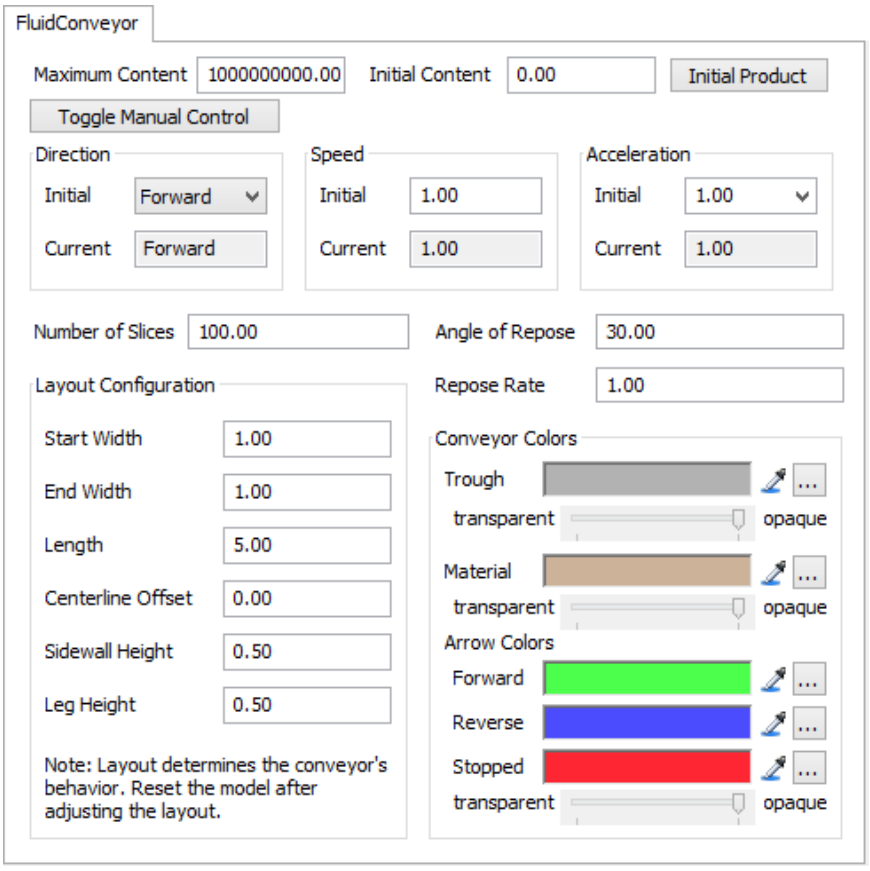
Maximum Port Rate - The maximum rate that material will leave this object through any one port.

Output port scale factor - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidBlender

FluidConveyor Page

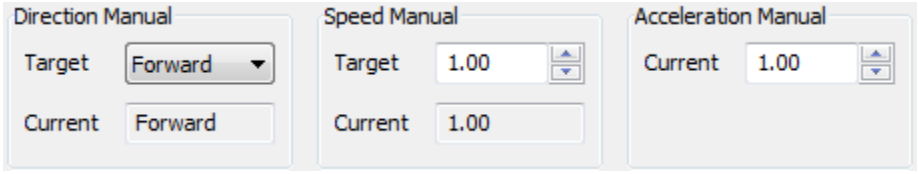


Maximum Content - The maximum amount of fluid material that this object can hold at any time.

Initial Content - The amount of material that is in the object when the model is reset.

Initial Product - This opens the Initial Product Window which allows the modeler to define the Product ID and sub-component mix of the material that is in this object.

Toggle Manual Control - Toggles manual speed control. You can change the direction, target speed and acceleration while the model is running. The manual speed control is designed to help you understand more about how the conveyor will react to changes in speed or direction.



Direction - Specifies the initial direction of the conveyor. The current direction is also displayed while the model is running.

Speed - Specifies the initial speed of the conveyor in the given initial direction. Speed values for the fluid conveyor cannot be negative. The current speed is also displayed while the model is running.

Acceleration - Specifies the initial acceleration of the conveyor. Infinite acceleration is defined as 0. The current acceleration is also displayed while the model is running.

Number of Slices - The number of slices of fluid material that are placed along the length of the conveyor. The more slices, the better the resolution for displaying the volume of fluid. However, the more slices in the fluid conveyor, the more computations, causing your model to run slower.

Angle of Repose - Defines the material's steepest angle of descent of the slope relative to the horizontal plane. This angle ranges between 0 and 90 degrees.

Repose Rate - The repose rate defines how quickly a reposing pile of material will reach its natural resting state (based on the angle of repose). A value of 0 will cause the Angle of Repose to be ignored. The

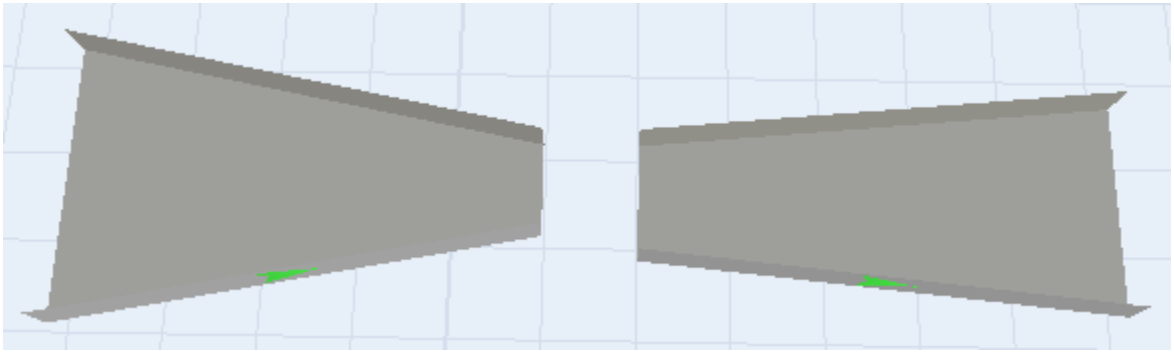
repose subroutine will be run the number of times specified in this field (the larger the number, the more processing time it well take each tick te repose).

Layout Configuration

The layout of the fluid conveyor affects the conveyor's behavior, so the model should be reset after the layout has been changed to apply the changes.

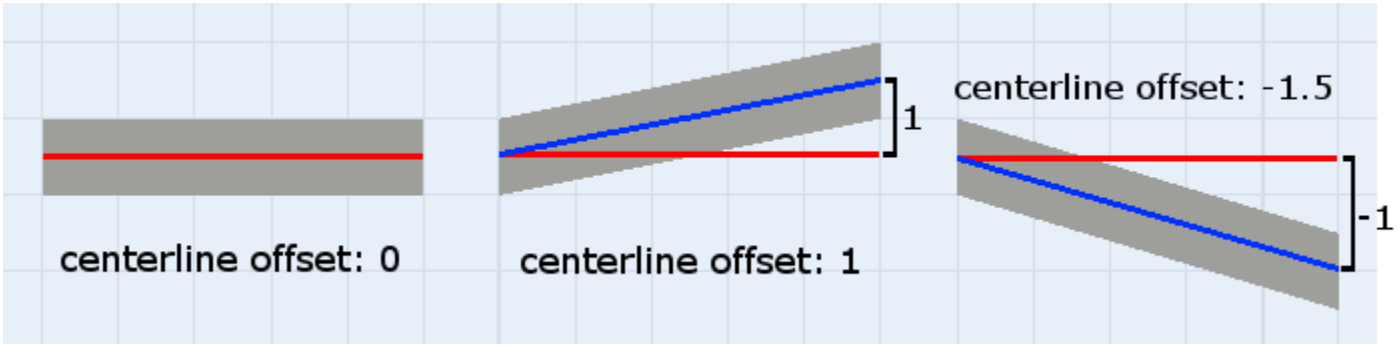
Start Width - Specifies the width of the start of the conveyor

End Width - Specifies the width of the end of the conveyor



Length - Specifies the length, or X dimension of the conveyor.

Centerline Offset - The centerline offset skews the conveyor's trough. The value specifies the distance and direction (can be positive or negative) that the centerline of the trough is offset from the standard centerline



Sidewall Height - Specifies the sidewall height of the conveyor's trough. This value is purely visual and has no effect on the behavior of the fluid conveyor.

Leg Height - Specifies the leg height of the conveyor. This value is purely visual and has no effect on the behavior of the fluid conveyor.

Conveyor Colors

Use the or to Sample a color or press "..." to choose a color.

Trough - Sets the color of the trough and legs. You may also change the transparency of the trough (allowing you to see the material's height profile through the sidewall).

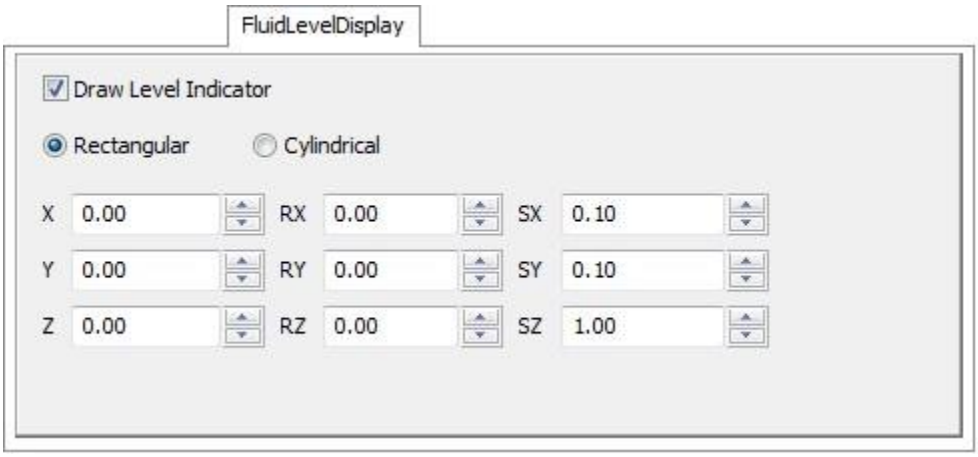
Material - Sets the color of the material. You may also change the transparency.

Arrow Colors - Set the color of the three directions of the fluid conveyor, Forward, Reverse, and Stopped. You may also change the transparency.

This Page is Used By

Fluid Conveyor

FluidLevelDisplay Page



Draw Level Indicator - If this is checked, the level indicator bar will be drawn on the object.

Rectangular - If this is selected, the level indicator bar will be drawn as a colored box.

Cylindrical - If this is selected, the level indicator bar will be drawn as a colored cylinder.

X - The X location of the bar.

Y - The Y location of the bar.

Z - The Z location of the bar.

RX - The rotation of the bar around the X axis.

RY - The rotation of the bar around the Y axis.

RZ - The rotation of the bar around the Z axis.

SX - The size of the bar in the X direction.

SY - The size of the bar in the Y direction.

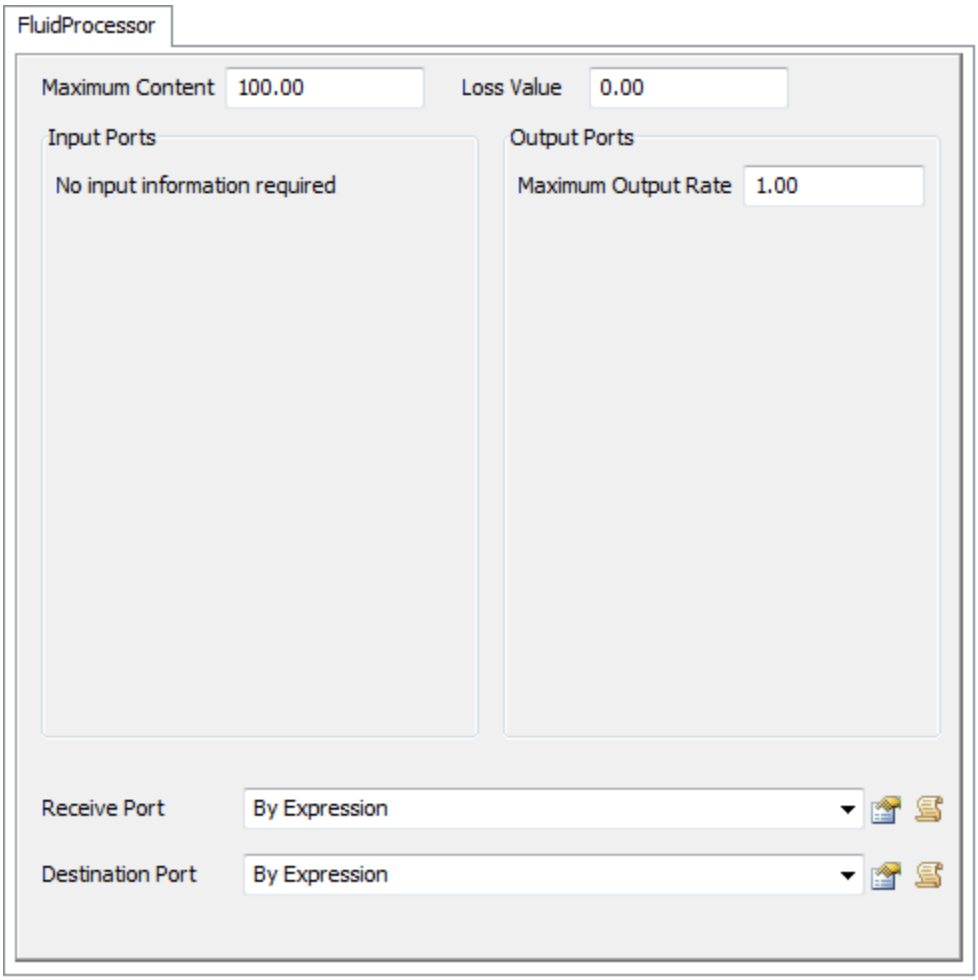
SZ - The size of the bar in the Z direction.

Note: The location and size values are expressed as a percentage (1.0 being 100%) of the size of the object.

This Page is Used By

FluidBlender
FluidGenerator
FluidMixer
FluidProcessor
FluidSplitter
FluidTank
FluidToItem
ItemToFluid

FluidProcessor Page



Maximum Content - The maximum amount of material that this object can hold.

Loss Amount - A value between 0 and 1 that represents the percentage of material that is lost going through the Processor. This loss could be due to evaporation, inefficiency or many other factors. A value of 0 means that there is no material lost, a value of 1 means that all material is lost. This loss is applied as soon as material is pulled into the Processor.

Receive Port - If this field returns a 0, the Processor will receive material from all input ports. If it returns a number greater than zero, the Processor will only receive material from that input port.

Destination Port - If this field returns a 0, the Processor will allow material to leave from all of its output ports. If it returns a number greater than zero, the Processor will only allow material out that output port.

Input Ports

There is no input information the modeler has to define.

Output Ports

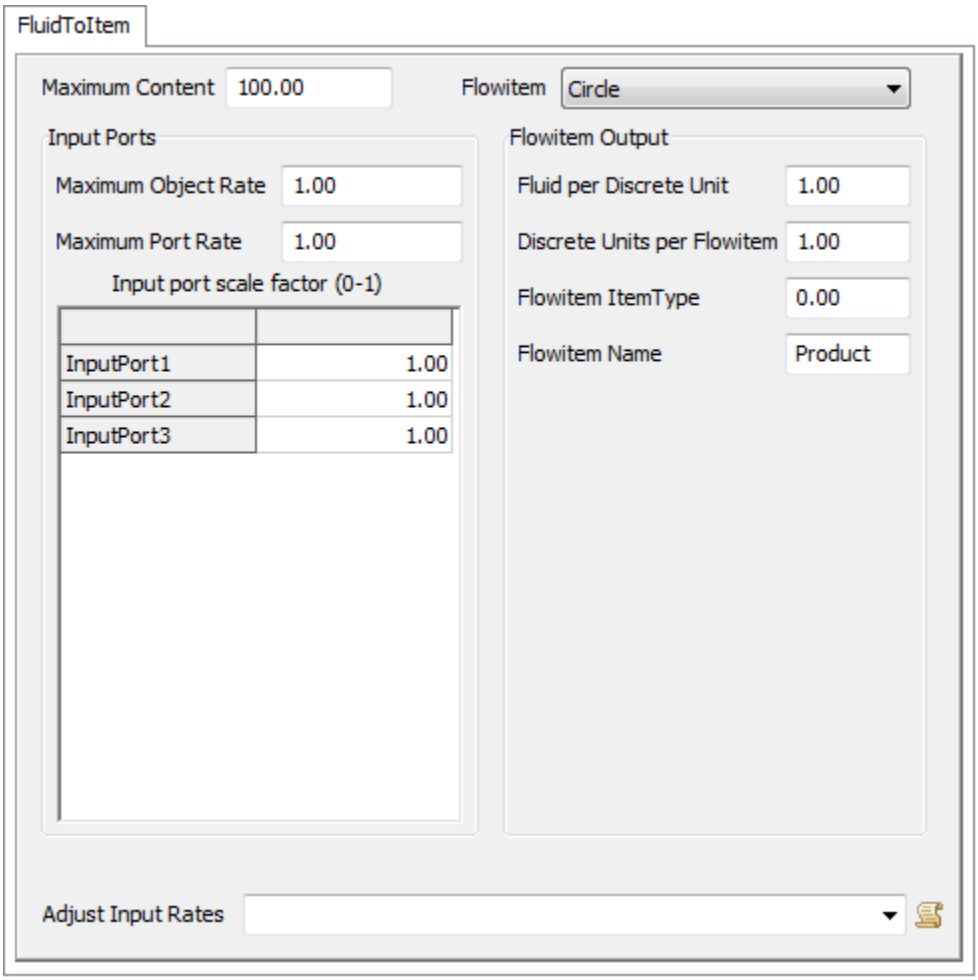
These properties affect how the object sends material to downstream objects.

Maximum Output Rate - The maximum rate that material will leave this object through all of the output ports combined. The actual rate will be determined by the rate of material coming into the Processor.

This Page is Used By

FluidProcessor

FluidToItem Page



Maximum Content - The maximum amount of fluid material that this object can hold at any time.

Flowitem - This is the class of flowitem that the FluidToItem will create.

Adjust Input Rates - This function is called every tick and allows the user to change the input rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Input Ports

These properties affect how the object receives material from upstream objects.

Maximum Object Rate - The maximum rate that material will enter this object through all output ports combined.

Maximum Port Rate - The maximum rate that material will enter this object through any one port.

Input port scale factor - Each row of the table is the scale factor for a single input port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual input ports during a model run.

Flowitem Output

These properties define when the FluidToItem creates a flowitem and some information that will be defined on the flowitem when it is created.

Fluid per Discrete Unit - This is the number of units of fluid material that are in a single discrete unit in the flowitem. For example: 5 gallons per can.

Discrete Units per Flowitem - This is the number of discrete units of material that are in each flowitem. For example: 10 cans per case, where a flowitem is a single case.

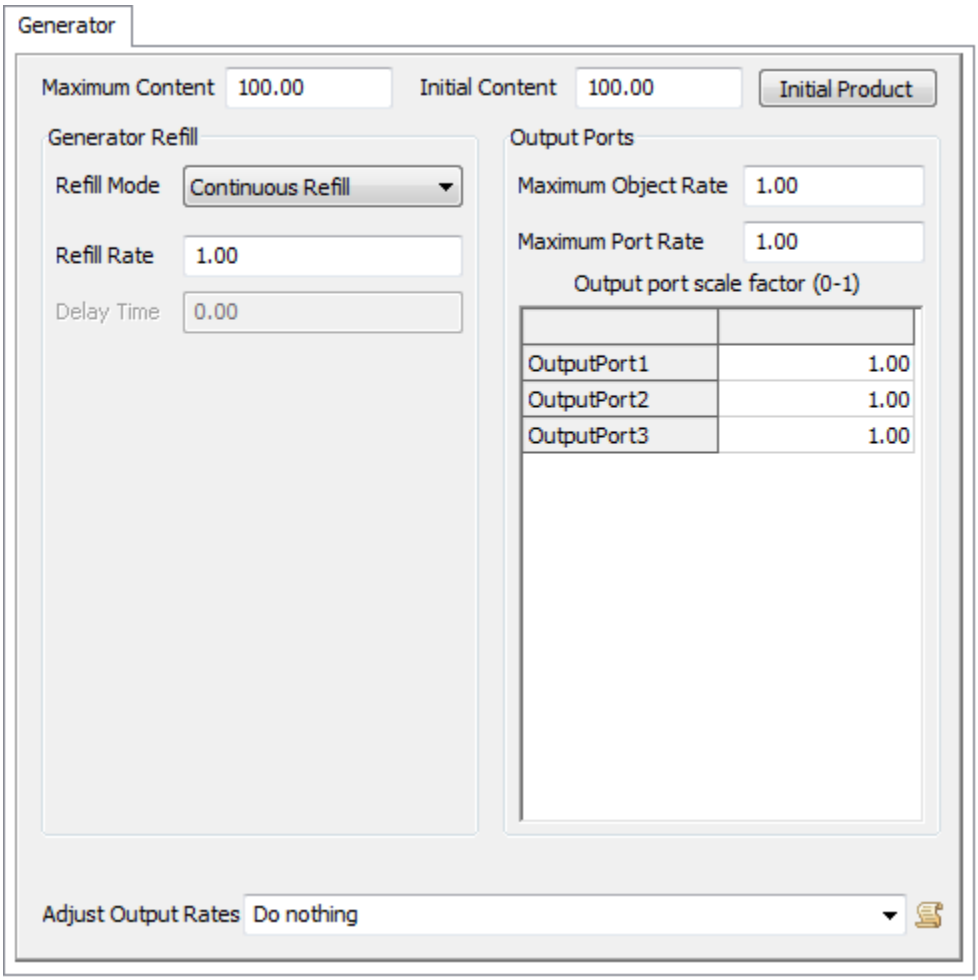
Flowitem ItemType - This value will be assigned to the itemtype of the flowitems as they are created. It can be changed using the OnCreation or OnExit triggers.

Flowitem Name - This name will be assigned to the flowitems that are created. It can be changed later using triggers.

This Page is Used By

FluidToItem

Generator Page



Maximum Content -The maximum amount of fluid material that this object can hold at any time.

Initial Content - The amount of material that is in the object when the model is reset.

Initial Product - This opens the Initial Product Window which allows the modeler to define the Product ID and sub-component mix of the material that is in this object.

Adjust Output Rates - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Generator Refill

These properties affect how the object refills itself as the model runs.

Refill Mode - This selects the type of refill the Generator performs. It can refill itself continuously (at a specified rate) or it can refill itself completely after it becomes empty.

Refill Rate - The rate at which the Generator refills itself. This is available if Continuous Refill is selected in the Refill Mode drop-down list.

Delay Time - The time that the Generator waits after becoming empty before it completely refills itself. This is available if Refill When Empty is selected in the Refill Mode drop-down list.

Output Ports

These properties affect how the object sends material to downstream objects.

Maximum Object Rate - The maximum rate that material will leave this object through all output ports combined.

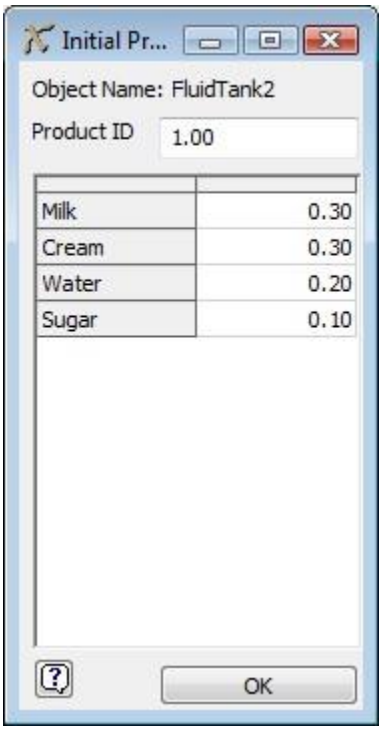
Maximum Port Rate - The maximum rate that material will leave this object through any one port.

Output port scale factor - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidGenerator

Initial Product Window



This window is used to define the initial Product ID and sub-component list for fluid material that is created by the object.

Object Name - The object that is creating this material.

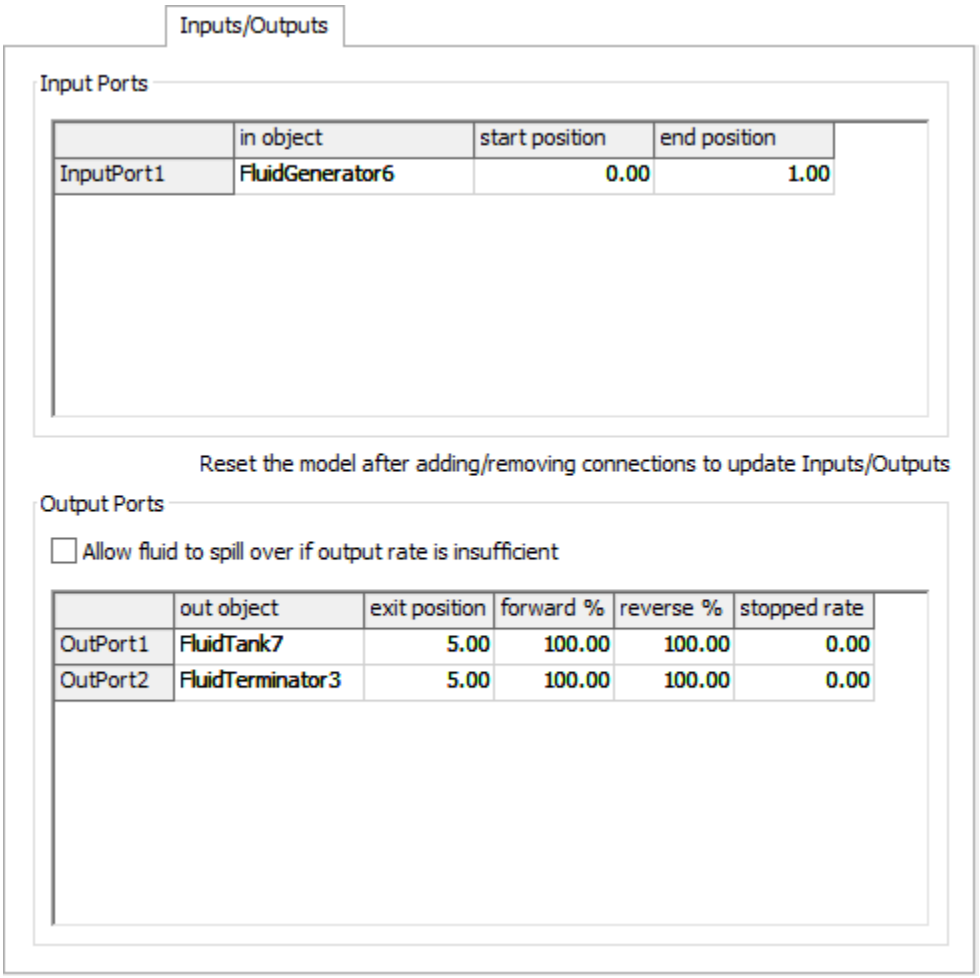
Product ID - The Product ID that will be assigned to the material that is created.

Sub-Component List - Each row in the list is a different sub-component that is available for this material. The list is defined on the FluidTicker Properties tab. The values are percentages from 0 to 1. They should add up to 1. All of the available sub-components are listed here, but the material does not have to use them all. If there are any that it does not use, the values in those rows should be set to 0.

This Page is Used By

FluidGenerator
FluidTank
ItemToFluid

Inputs/Outputs Page



Input Ports Table

Displays all of the objects currently connected to an input port of the Fluid Conveyor.

In Object - Displays the name of the input object.

Start Position - The start point of the input. This value is in the model's length units and must be greater than or equal to 0 and less than the end position.

End Position - The end point of the input. This value is in the model's length units and must be less than or equal to the length of the Fluid Conveyor and greater than the start position.

Allow Spillage - If this is checked, fluid will be allowed to spill at outputs if the downstream object cannot take as much fluid as the Fluid Conveyor is sending. Any extra fluid left at the end of the conveyor will also spill. Total spillage is tracked each tick.

Output Ports Table

Displays all of the objects currently connected to an output port of the Fluid Conveyor.

Out Object - Displays the name of the output object.

Exit Position - The exit point of the output. This value is in the model's length units and must be greater than or equal to 0 and less than the end position.

Forward % - This specifies the percentage of fluid that should exit at this output when the Fluid Conveyor is moving forward. The Fluid Conveyor will attempt to send the specified percentage of fluid through the output. If the downstream object cannot handle the total amount of fluid, it will pass the output, unless spillage is allowed, then the excess will spill onto the floor

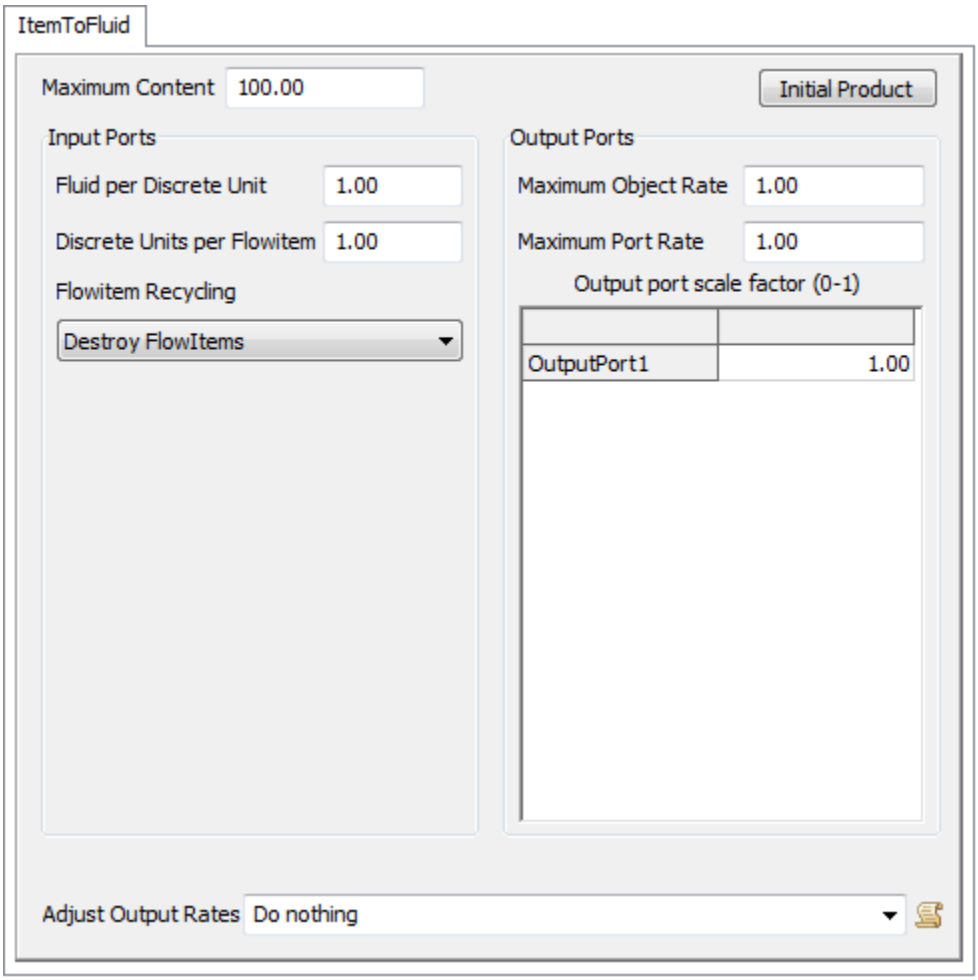
Reverse % - This specifies the percentage of fluid that should exit at this output when the Fluid Conveyor is moving forward. The Fluid Conveyor will attempt to send the specified percentage of fluid through the output. If the downstream object cannot handle the total amount of fluid, it will pass the output, unless spillage is allowed, then the excess will spill onto the floor

Stopped Rate - The stopped rate specifies how much fluid should leave through the output when the Fluid Conveyor is stopped. For example, if the stopped rate is 0.1 and the material sitting on top of an output is 0.5, the output will take 0.1 fluid units per tick, until there is no more fluid available to take.

This Page is Used By

Fluid Conveyor

ItemToFluid Page



Maximum Content - The maximum amount of fluid material that this object can hold at any time.

Initial Product - This opens the Initial Product Window which that allows the modeler to define the ProductID and sub-component mix of the material that is created by this object.

Adjust Output Rates - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Input Ports

Fluid per Discrete Unit - This is the number of units of fluid material that are in a single discrete unit in the flowitem. For example: 10 pounds per bag.

Discrete Units per Flowitem - This is the number of discrete units of material that are in each flowitem. For example: 5 bags per pallet, where a flowitem is a single pallet.

Flowitem Recycling - The modeler uses this drop-down list to decide where to store flowitems that need to be recycled. They should send flowitems back to the section of the flowitem bin that they originally came from.

Output Ports

These properties affect how the object sends material to downstream objects.

Maximum Object Rate - The maximum rate that material will leave this object through all output ports combined.

Maximum Port Rate - The maximum rate that material will leave this object through any one port.

Output port scale factor - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual output ports during a model run.

This Page is Used By

ItemToFluid

Marks Page

Low Mark - If the content passes this value (while rising or falling), the PassingLowMark Trigger will fire.

Mid Mark - If the content passes this value (while rising or falling), the PassingMidMark Trigger will fire.

High Mark - If the content passes this value (while rising or falling), the PassingHighMark Trigger will fire.

Passing Low Mark - If the content passes the Low Mark, this trigger fires. Its common uses include opening and closing ports or sending messages. There is an access variable that informs the modeler if the fluid level is rising through the mark or falling.

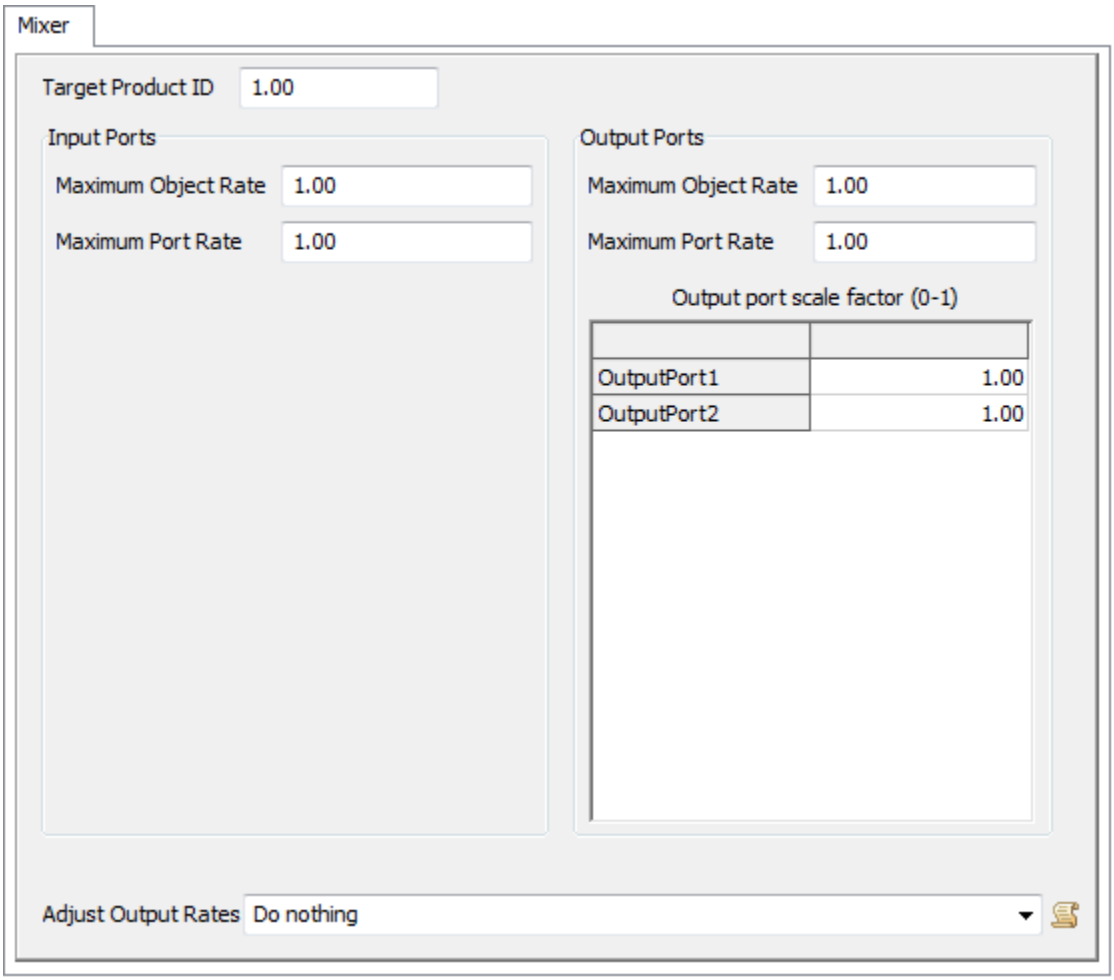
Passing Mid Mark - If the content passes the Mid Mark, this trigger fires. Its common uses include opening and closing ports or sending messages. There is an access variable that informs the modeler if the fluid level is rising through the mark or falling.

Passing High Mark - If the content passes the High Mark, this trigger fires. Its common uses include opening and closing ports or sending messages. There is an access variable that informs the modeler if the fluid level is rising through the mark or falling.

This Page is Used By

FluidTank

Mixer Page



Target Product ID - The ProductID that will be assigned to the material that leaves this object.

Adjust Output Rates - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Input Ports

These properties affect how the object receives material from upstream objects.

Maximum Object Rate - The maximum rate that material will be allowed into this object through all input ports combined. The actual input rate is based on the amount of material available upstream and the space available in this object.

Maximum Port Rate - The maximum rate that material will be allowed into this object through any single input port.

Output Ports

These properties affect how the object sends material to downstream objects.

Maximum Object Rate - The maximum rate that material will leave this object through all output ports combined.

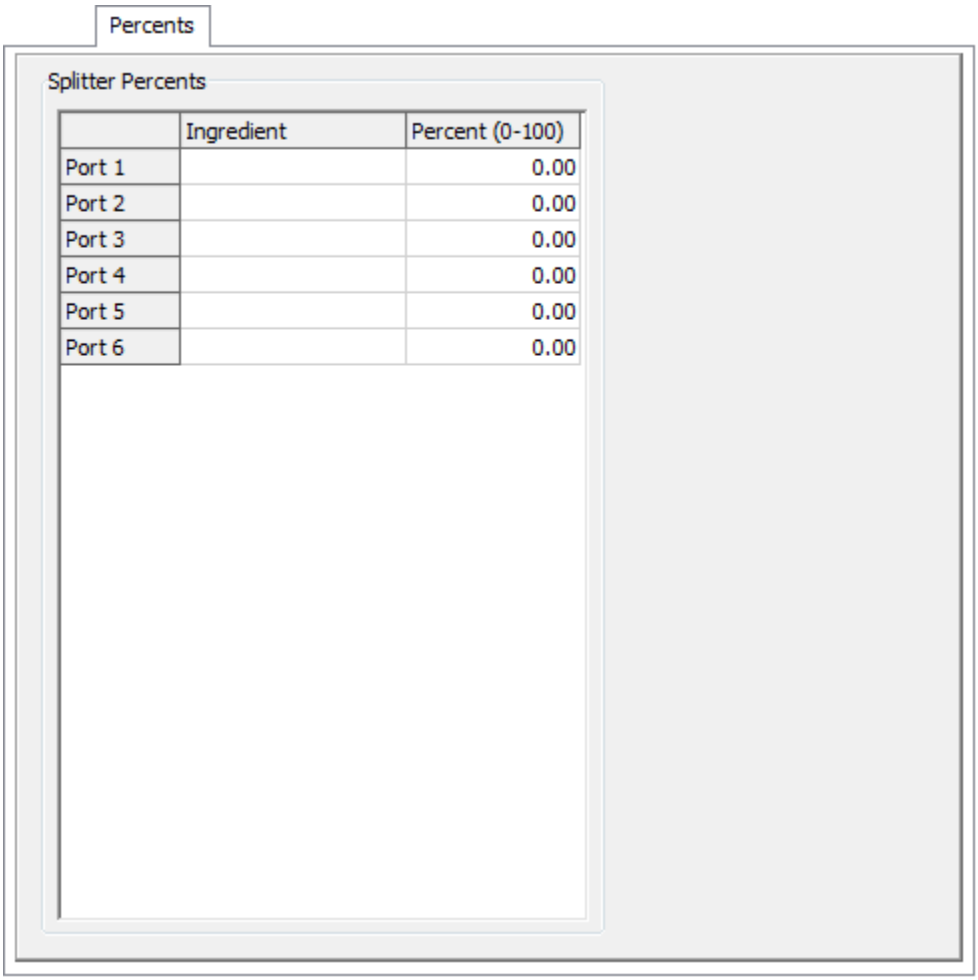
Maximum Port Rate - The maximum rate that material will leave this object through any one port.

Output port scale factor - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidMixer

Percents Page



Splitter Percents

Each row of this table represents a single output port. The rows do not appear in the table unless the object is already connected to downstream objects when this window is opened. There are two columns that the modeler can change in the table:

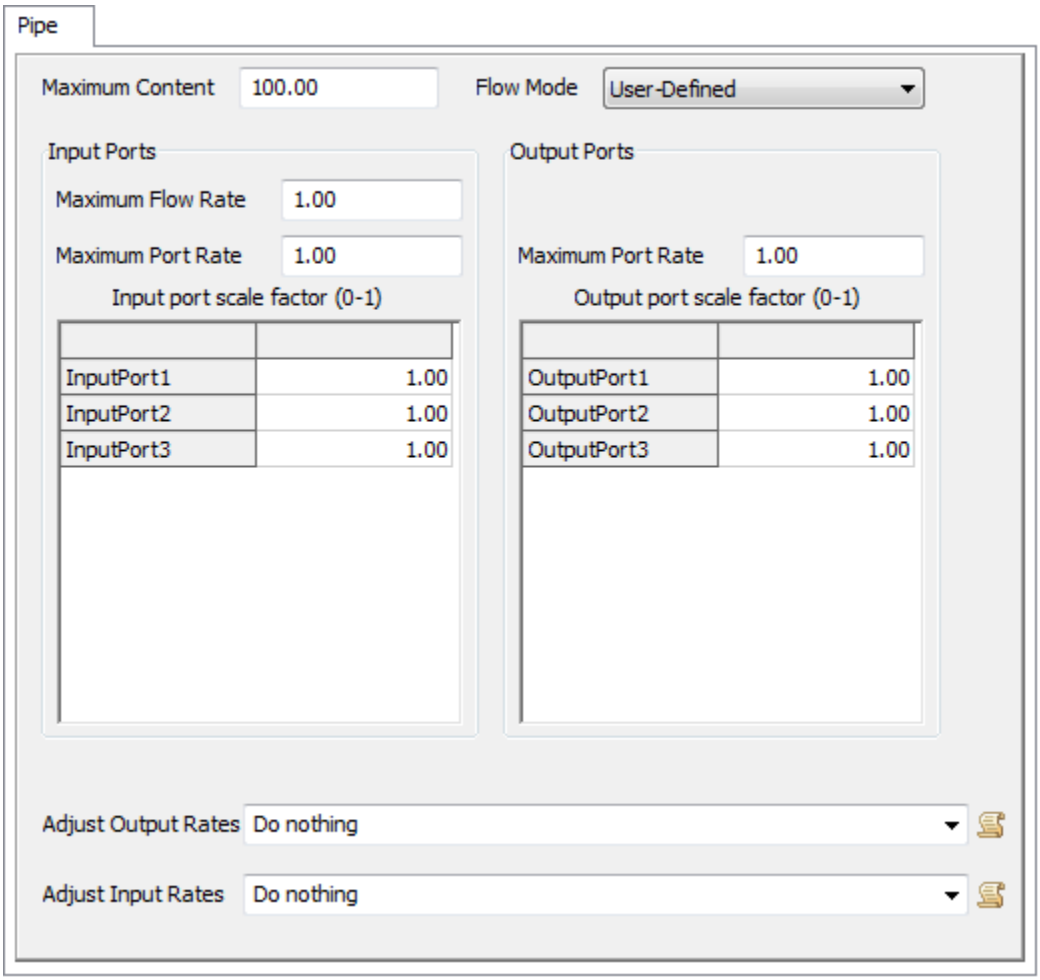
Ingredient - This is a text description of the material going to the port the row represents. This is for the modeler's use only, the Splitter will ignore this value.

Percent - This is a number between 0 and 100 that is the percentage of the total outgoing material that should go to the port represented by the row. The Splitter will adjust the actual amount of material sent to each port to make sure these percentages are correct, even when there is not enough material or space available to send at the maximum rate.

This Page is Used By

FluidSplitter

Pipe Page



Maximum Content - The maximum amount of material that this object can hold.

Flow Mode - The Pipe has three different modes that can be used to define how fluid is sent downstream:

Flow Evenly - The output ports are configured to have a maximum flow rate equal to the incoming flow rate divided by the number of output ports. The output ports may not send the same amount, depending on the content of the downstream objects

First Available - The output ports are configured to have a maximum flow rate equal to the incoming flow rate. Material will be sent to downstream objects in a first-come-first-served manner.

User Defined - The modeler has control over the input rate (both for the object and the individual ports) and the output rate for individual ports.

Adjust Input Rates - This function is called every tick and allows the user to change the input rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors, getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Adjust Output Rates - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Input Ports

These properties affect how the object receives material from upstream objects.

Maximum Flow Rate - The maximum rate that material will be allowed into this object through all input ports combined. This value serves as both the maximum input and maximum output rates. The actual rate is based on the amount of material available upstream and the space available in this object. Material will attempt to leave the Pipe at the same rate that it came in. If there is not enough room downstream, the material will "back up" and more (up to the maximum rate) will be available to send in the next tick.

Maximum Port Rate - The maximum rate that material will be allowed into this object through any single input port.

Input port scale factor - Each row of the table is the scale factor for a single input port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual output ports during a model run.

Output Ports

These properties affect how the object sends material to downstream objects.

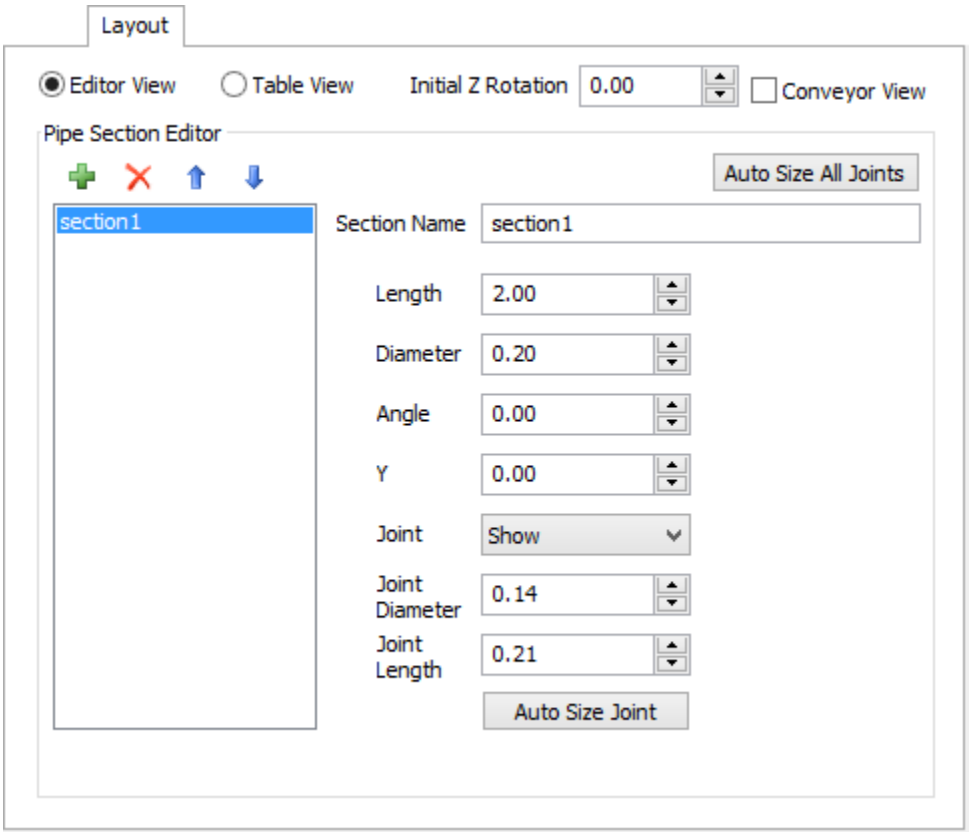
Maximum Port Rate - The maximum rate that material will leave this object through any one port.

Output port scale factor - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidPipe

Pipe Layout Page



Editor View - Displays the Pipe Layout editor view as shown above.

Table View - Displays the Pipe Layout table view as shown below.

Initial Z Rotation - This is the rotation around the Z axis that is applied to the starting point of the Pipe. This is used to orient the Pipe in a particular direction before sections are drawn.

Conveyor View - If this box is checked, the Pipe will be drawn as a simple conveyor. This does not change the functionality of the pipe. See the Fluid Conveyor object for conveyor functionality.

Pipe Section Editor

- Adds a new sensor to the table.

- Removes the selected sensor from the table, or if no sensor is selected, removes the last sensor in the table.

- Moves the selected sensor up in the list.

- Moves the selected sensor down in the list.

Section Name - The name of the pipe section. This is purely for the modeler's convenience and has no affect on the model.

Length - The length of the section

Diameter - The diameter of this section of the Pipe. If the Pipe is being shown as a conveyor, this is the width of the end of the section. The actual section will become wider or more narrow depending on the diameter value of the previous section.

Z Rotation - The rotation around the Z Axis that is applied at the end of the section.

Y Rotation - The rotation around the Y Axis that is applied at the end of the section.

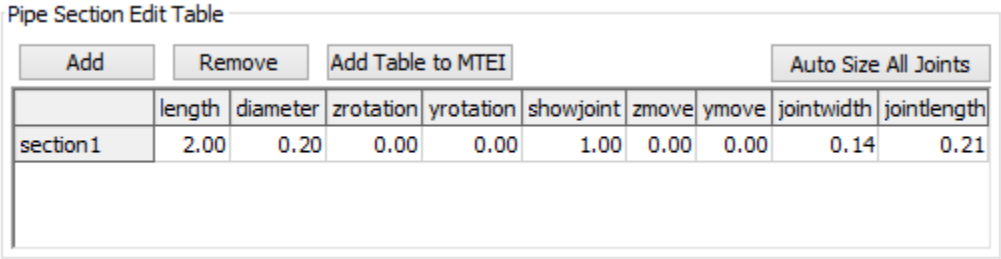
Show Joint - Specifies whether the pipe's joint will be drawn between the end of this section and the start of the next one. This value is ignored if the Pipe is being drawn as a Conveyor.

Join Diameter - Specifies the diameter of the joint.

Joint Length - Specifies the length of the joint.

Auto Size Joint - Automatically adjusts the joint size based on the diameter and rotation of the pipe sections.

Pipe Section Edit Table



Add - Adds a new pipe section by copying the selected section.

Remove - Removes the selected pipe section.

Add Table to MTEI - This buttons adds the table as a row in the Multiple Table Excel Import.

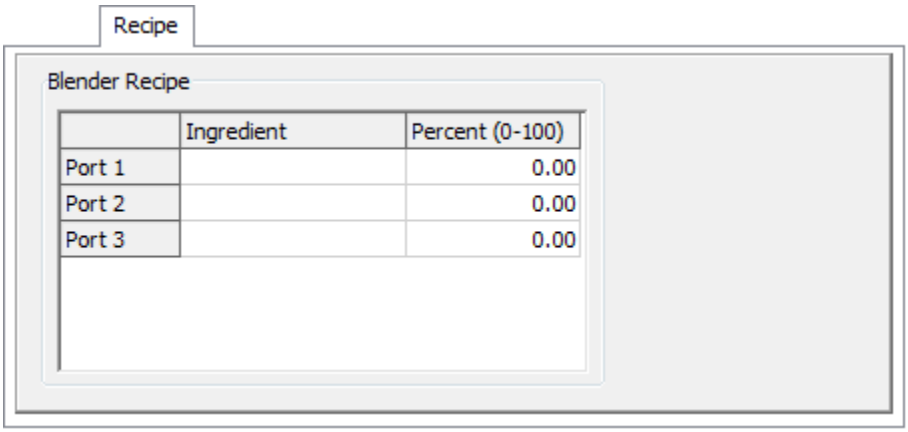
Section Table - Each row in the table represents a single section of the Pipe. The columns are described above.

Note: The zmove and ymove columns should not be changed. They are used by FlexSim.

This Page is Used By

FluidPipe

Recipe Page



Blender Recipe

Each row of this table represents a single input port. The rows do not appear in the table unless the object is already connected to upstream objects when the Properties GUI is opened. There are two columns that the modeler can change in the table:

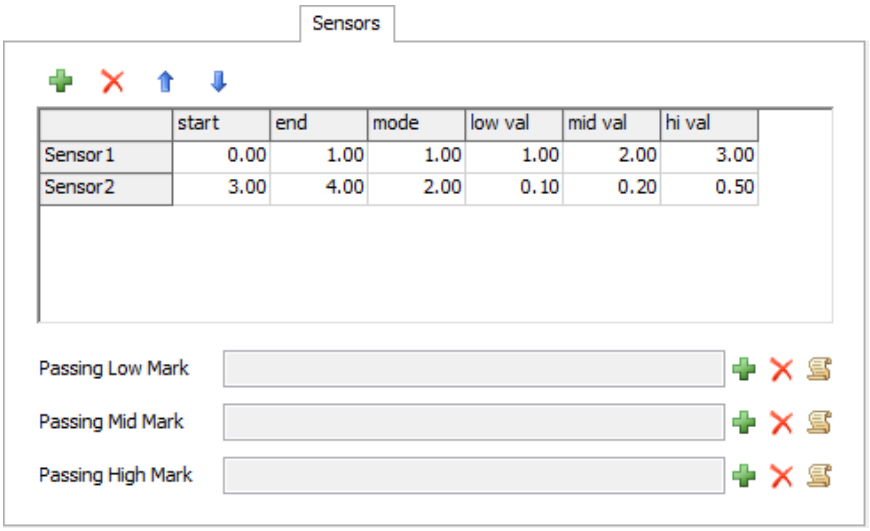
Ingredient - This is a text description of the ingredient coming from the port the row represents. This is for the modeler's use only, the Blender will ignore this value.

Percent - This is a number between 0 and 100 that is the percentage of the total incoming material that should come from the port represented by the row. The Blender will adjust the actual amount of material pulled from each port to make sure these percentages are correct, even when there is not enough material or space available to pull at the maximum rate.

This Page is Used By

FluidBlender

Sensors Page



Sensor Table

Displays all of the sensors for the Fluid Conveyor.

- Adds a new sensor to the table.

- Removes the selected sensor from the table, or if no sensor is selected, removes the last sensor in the table.

- Moves the selected sensor up in the list.

- Moves the selected sensor down in the list.

Start - The starting point of the sensor. This value is in the model's length units and must be greater than or equal to 0 and less than the end point.

End - The end point of the sensor. This value is in the model's length units and must be less than or equal to the total length of the conveyor and greater than the start point.

Mode - The sensor has two modes, Volume(1) and Peak Height(2). Volume will look at the total volume between the starting and ending point. Peak Height will look at the highest point between the starting and ending point.

Low, Mid, High Val - These values specify the low, mid, and high points for the volume or peak height that, when crossed, will trip the sensor. When the volume or peak height of material in a sensor range rises or falls through any of these three values, one of the three passing triggers will fire.

Sensor Triggers

Passing Low Mark - This trigger is fired when fluid volume or peak height rises or palls through the defined Low Val.

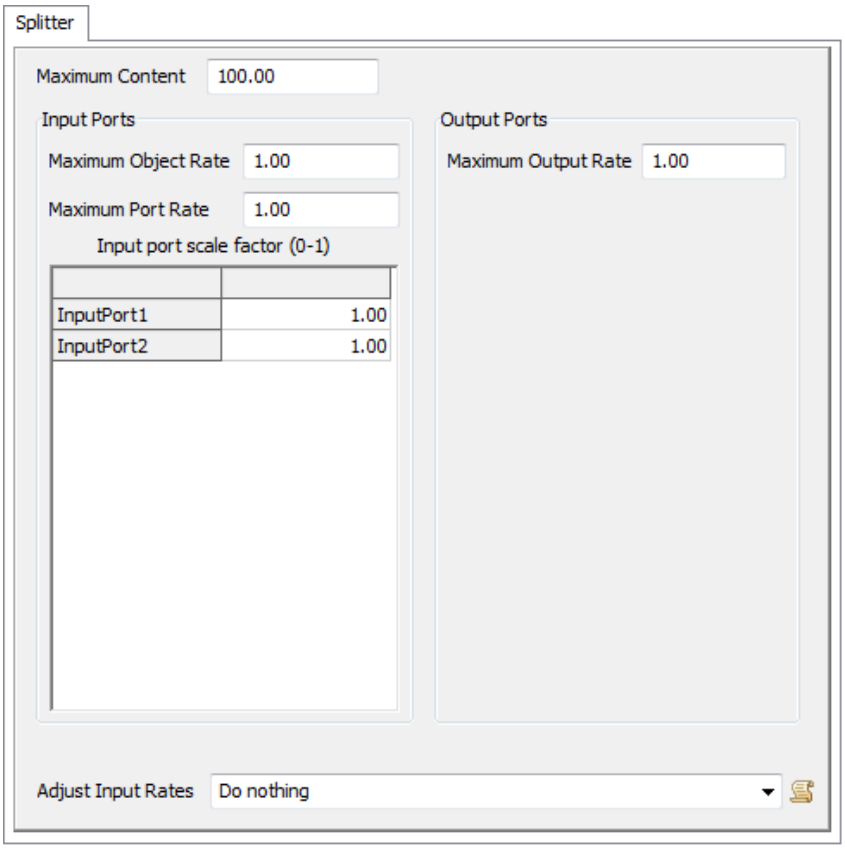
Passing Mid Mark - This trigger is fired when fluid volume or peak height rises or palls through the defined Mid Val.

Passing High Mark - This trigger is fired when fluid volume or peak height rises or palls through the defined High Val.

This Page is Used By

Fluid Conveyor

Splitter Page



Maximum Content - The maximum amount of fluid material that this object can hold at any time.

Adjust Input Rates - This function is called every tick and allows the user to change the input rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Input Ports

These properties affect how the object receives material from upstream objects.

Maximum Object Rate - The maximum rate that material will enter this object through all output ports combined.

Maximum Port Rate - The maximum rate that material will enter this object through any one port.

Input port scale factor - Each row of the table is the scale factor for a single input port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual input ports during a model run.

Output Ports

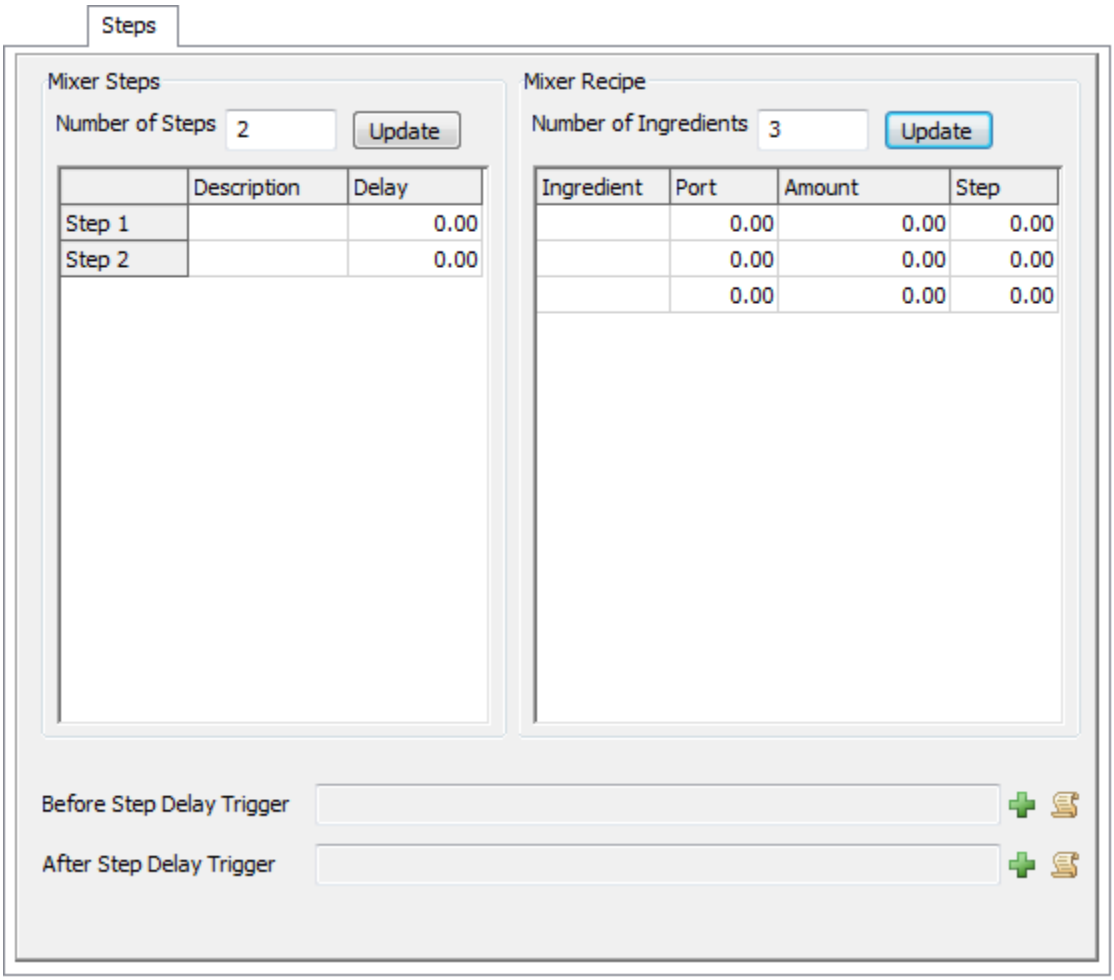
These properties affect how the object sends material to downstream objects.

Maximum Output Rate - The maximum rate that material will leave this object through all output ports combined.

This Page is Used By

FluidSplitter

Steps Page



Before Step Delay Trigger - This trigger fires after all of the material for a step has been collected, but before the step's delay time begins. This gives the modeler a chance to do things like call an operator for the delay.

After Step Delay Trigger - This trigger fires after the delay for a step is complete. It gives the modeler a chance release an operator or send messages to other objects.

Mixer Steps

Number of Steps - This is the number of steps that the Mixer will go through for every batch of material that it makes.

Update - Pressing this button updates the Step Table so that it has the number of rows specified by the modeler.

The Step Table shows all of the steps that the Mixer must go through for each batch. Each step has two columns that the modeler must fill out:

Description - This is a text description of the step. It is displayed by the Mixer's name in the model view window when the Mixer is on the step.

Delay - This is the amount of time that the Mixer must wait after collected all of the ingredients for the step before it can go on to the next step.

Mixer Recipe

Number of Ingredients - This is the number of ingredients that the Mixer will pull as it goes through its Step Table.

Update - Pressing this button updates the Ingredients Table so that it has the number of rows specified by the modeler.

The Ingredients Table show all of the ingredients that the Mixer pulls as it goes through its Step Table. If a single ingredient needs to be pulled in more than one step, it should appear in more than one row in the table. The table has four columns that the modeler must fill out:

Ingredient - This is a text description of the ingredient that the row represents. It is only to help the modeler document their model. It does not affect the Mixer's behavior.

Port - This is the input port that the ingredient will be pulled from.

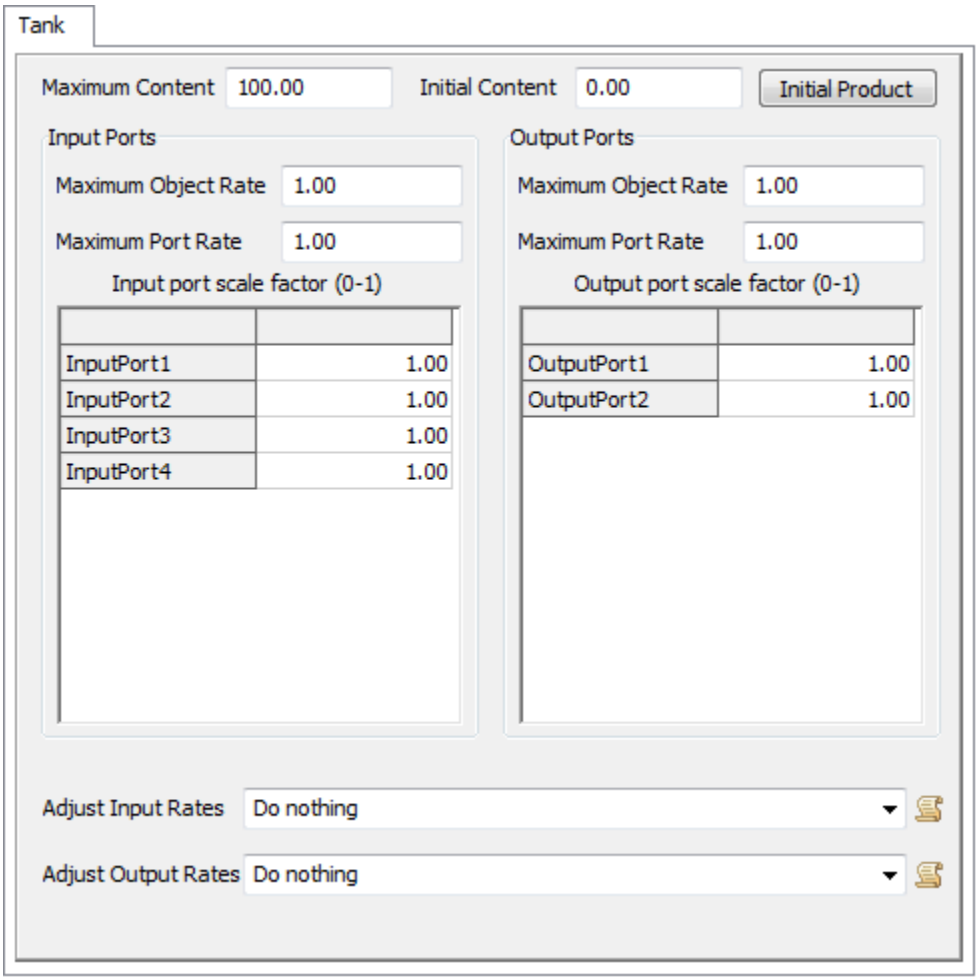
Amount - This is the amount of the ingredient that will be pulled.

Step - This is the step number that the Mixer must be in for this ingredient to be pulled.

This Page is Used By

FluidMixer

Tank Page



Maximum Content - The maximum amount of fluid material that this object can hold at any time.

Initial Content - The amount of material that is in the object when the model is reset.

Initial Product - This opens the Initial Product Window which that allows the modeler to define the Product ID and sub-component mix of the material that is in this object.

Adjust Input Rates - This function is called every tick and allows the user to change the input rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Adjust Output Rates - This function is called every tick and allows the user to change the output rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Input Ports

These properties affect how the object receives material from upstream objects.

Maximum Object Rate - The maximum rate that material will enter this object through all output ports combined.

Maximum Port Rate - The maximum rate that material will enter this object through any one port.

Input port scale factor - Each row of the table is the scale factor for a single input port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual input ports during a model run.

Output Ports

These properties affect how the object sends material to downstream objects.

Maximum Object Rate - The maximum rate that material will leave this object through all output ports combined.

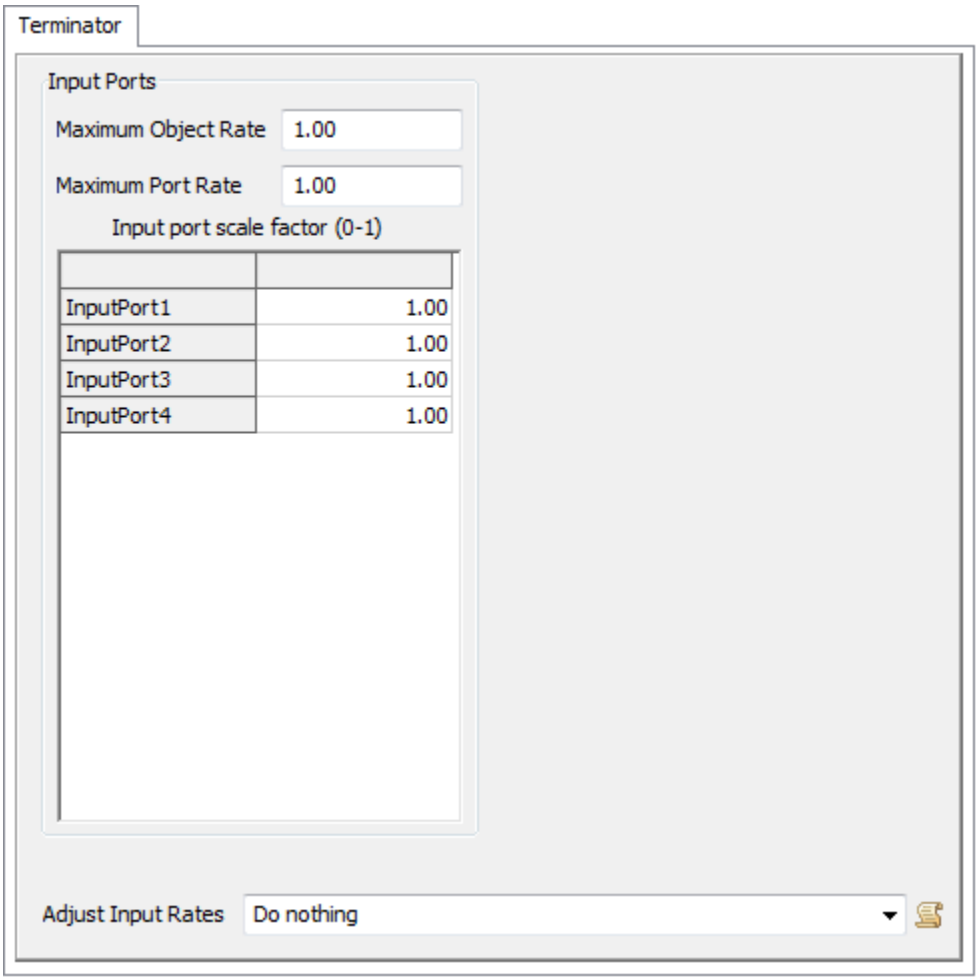
Maximum Port Rate - The maximum rate that material will leave this object through any one port.

Output port scale factor - Each row of the table is the scale factor for a single output port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual output ports during a model run.

This Page is Used By

FluidTank

Terminator Page



Adjust Input Rates - This function is called every tick and allows the user to change the input rates and scale factors during a model run. There are access variables that are references to the nodes that hold the rates and scale factors. To read the current rate (object or port) the getnodenum() command should be used. To change the rate the setnodenum() should be used. To read or change the scale factors getnodenum() and setnodenum() should be used in conjunction with the rank() command.

Input Ports

These properties affect how the object receives material from upstream objects.

Maximum Object Rate - The maximum rate that material will enter this object through all output ports combined.

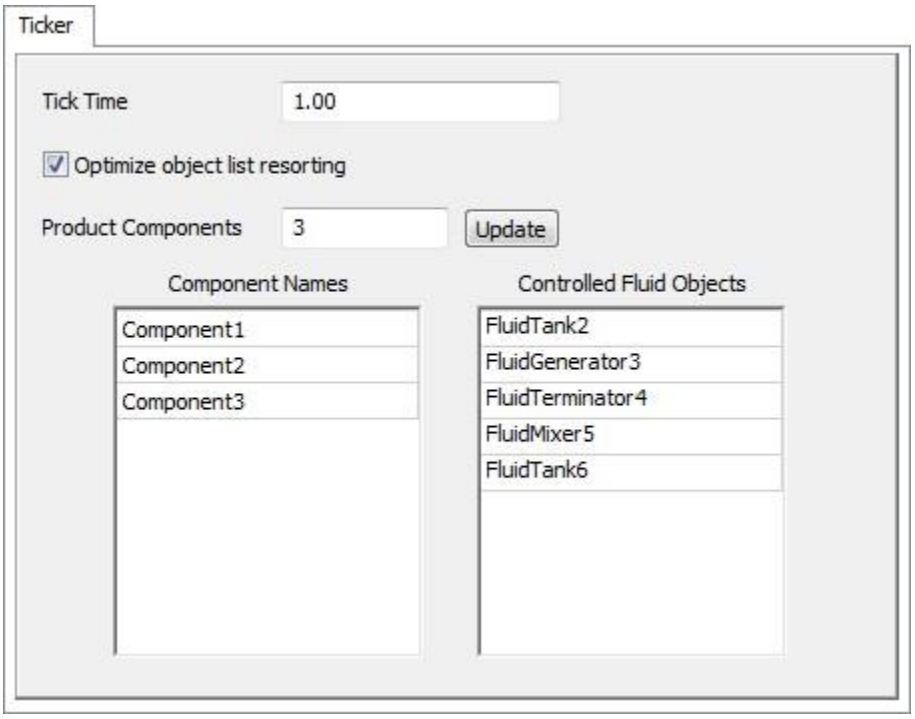
Maximum Port Rate - The maximum rate that material will enter this object through any one port.

Input port scale factor - Each row of the table is the scale factor for a single input port. The value is multiplied by the maximum port rate to determine the actual maximum rate for that specific port. This allows the modeler to change the rate of individual input ports during a model run.

This Page is Used By

FluidTerminator

Ticker Page



Tick Time - This is length of time in each tick. At the end of a tick, the Ticker calculates how much fluid moved between the fluid objects in the model.

Optimize object list resorting - The Ticker keeps an internal list of the order that the Fluid Objects should be evaluated. if this box is not checked, the order in which certain objects are evaluated may be different in different runs of the model. This can cause a model to give different results, even if nothing in the model has actually changed. Typically, this box should be checked.

Product Components - This is the number of sub-components available to all of the fluid objects in the model. All of the objects use the same list of sub-components, although they do not have to specify a value greater than 0 for all of the components.

Update - Pressing this button updates the list of component names so that there are the number that the modeler specified.

Component Names - This table lists the names of the sub-components that are available to all of the fluid objects in the model.

Controlled Fluid Objects - This table lists the names of the fluid objects that are controlled by this Ticker.

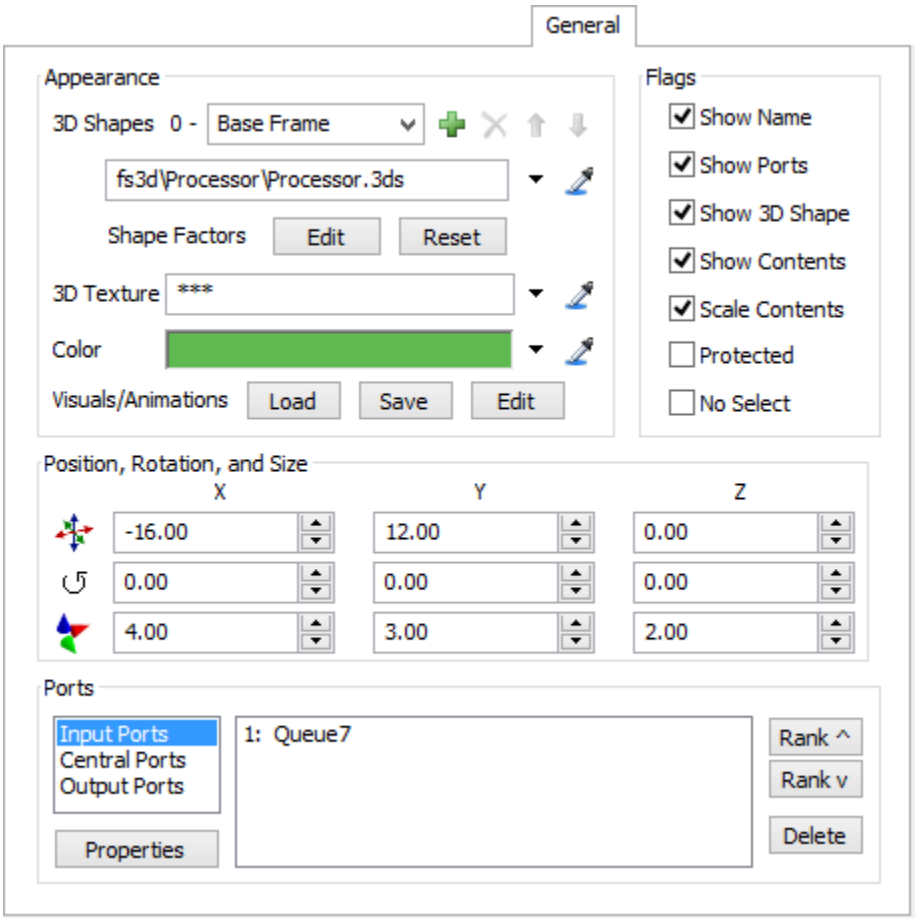
This Page is Used By

Ticker

Shared Properties Pages

1. General
2. Labels
3. Triggers
4. Statistics

General Page



Appearance

3D Shapes - This option specifies the 3D shape(s) for the object. The drop down contains all of the shape frames for the object. The number (0 -) is the shape frame index for use in the setframe() command. For more information, see the Shape Frames page. You can specify any of the following types of 3D files:

- .wrl, .3ds, .dxf, .stl, .skp, .dae, .obj, .ac, .x, .ase, .ply, .ms3d, .cob, .md5mesh, .irr, .irrmesh, .ter, .lxo, .csm, .scn, .q3o, .q3s, .raw, .off, .mdl, .hmp, .scn, .xgl, .zgl, .lvo, .lvs, .blend

3D shapes can either be referenced through the or loaded 3D shapes will display in the down arrow menu. You can also select Browse... from this menu to load a new shape.

- Adds a Shape Frame to the object.

- Removes the current Shape Frame.

- Moves the current Shape Frame up in the list, changing its frame index.

- Moves the current Shape Frame down in the list, changing its frame index.

Shape Factors - Click on this button to display the object's shape factors popup. To learn more about shape factors, see the Shape Factors page.

3D Texture - This field specifies the object's 3D texture. If the 3D shape does not already have a texture defined within its 3D file, then this texture will be drawn on the face of the 3D shape. Note that if the object's 3D shape already has a texture defined, then this texture will not be used.

Color - This field specifies the color of the object. Note that if the object already has materials defined in its 3D shape's file, then this color will not show. This color shows through only if no materials are defined in the 3D file.

Visuals/Animations

These buttons allow you to save/load visual information for objects. You can also access an object's animations through the 3D views right click menu.

Load/Save - These buttons load/save all of the **Appearance** settings of an object, allowing you to save shape, texture, color, and animations in a .t file to load into other objects you want to look the same.

Edit - This button opens the Animation Creator, which allows you to create animations for FlexSim Objects.

Flags

Here you can check different boxes to show/hide different parts of the object, such as the contents of the object, the name, the ports, etc.

These flags can also be toggled through the Edit Selected Objects view.

Show Name - Toggles displaying the object's name in the 3D view.

Show Ports - Toggles displaying port connections in the 3D view.

Show 3D Shape - Toggles displaying the object's 3D shape.

Show Contents - Toggles displaying the object's contents.

Scale Contents - If checked, any objects within the content of this object will be scaled according to the size of this object.

Protected - If checked, this object will not allow the user to move, size, rotate, or delete the object.

No Select - If checked, this object will not be able to be clicked on the 3D view. To gain access to an object with the No Select flag checked, find it in the Tree Window.

Position, Rotation, and Size

Here you can set the position, rotation, and size of the object based on X, Y, and Z values.

: Change the position of the object.

: Change the rotation of the object.

: Change the size of the object.

Ports

This area lets you edit the object's connections. Select either Input Ports, Central Ports, or Output Ports from the combobox on the left. The list on the right shows the appropriate connections. Once you have finished editing an object's connections, you will need to reset the model before running it again.

Rank ^ - This button will move the selected connection up in the list.

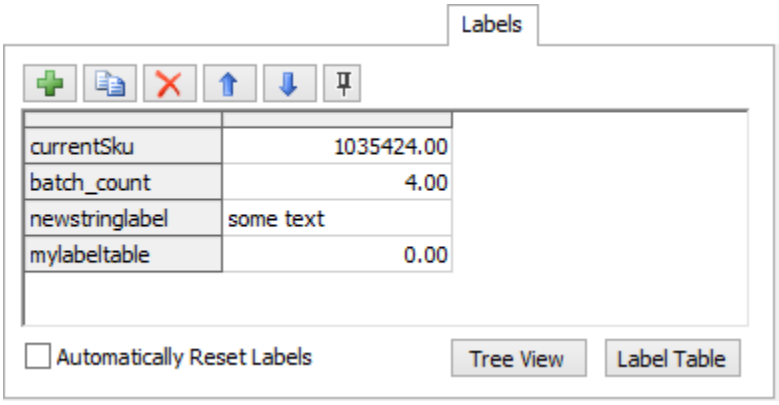
Rank v - This button will move the selected connection down in the list.

Delete - This button will delete the selected connection.

Properties - This button will open a new Properties window for the selected object.

Labels Page

Labels are custom variables that you can specify on the object. For example, if you want to keep track of the number of flowitems of itemtype 3 that have entered an object, you can use a label to keep a record of this value. Use the commands setlabelnum(), getlabelnum(), and label() to interact with labels you have created. More information on these commands is found in the command summary.



The main panel shows a list of the labels on this object.

- Adds a new label with number or string data to the end of the list of labels.

- Duplicates the selected label(s).

- Deletes the selected label(s).

- Moves the selected label(s) up or down in the list.

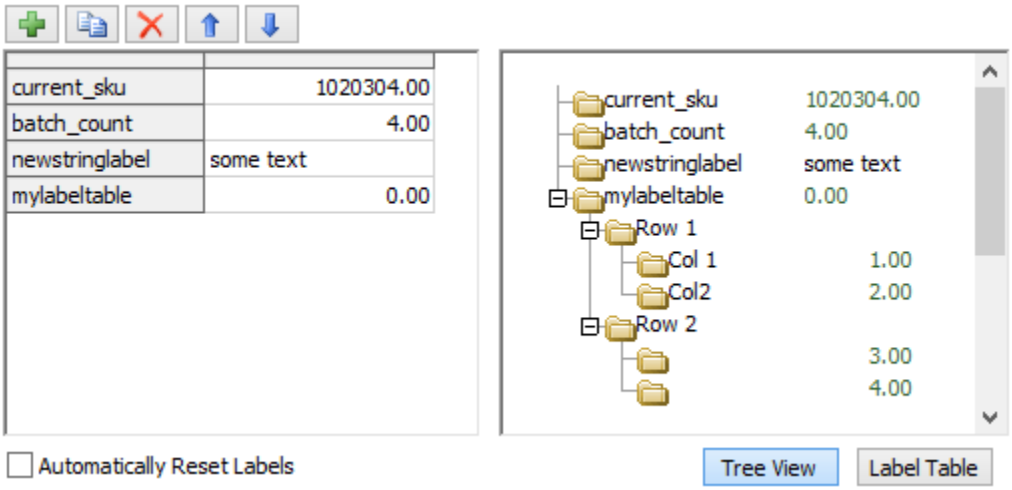
- Pins the selected label(s) (or all labels if there is no selection) to a Dashboard as either a table of values, bar chart or line graph. Note: Pinning labels with string data will display 0 for its value.

Automatically Reset Labels - If this option is checked, then the object will automatically reset its labels back to their initial values on reset. When you apply the window, the values shown will be saved as the reset values. The reset values will also be automatically set when directly editing the labels on this tab while the model is reset.

Labels can also be edited through the Quick Properties.

Tree View

If you click the Tree View button, you can view and edit the list of labels in a tree. The window then has a splitter bar in the middle to change the size of each panel. You can click the Tree View button again to hide to tree view panel.

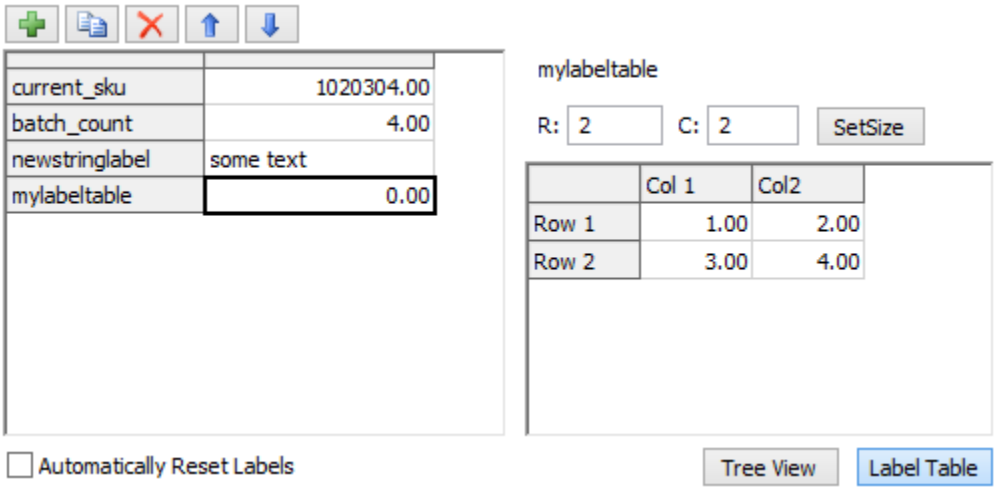


Label Table View

If you click the Label Table button, you can view and edit label tables within the same view. The window then has a splitter bar in the middle to change the size of each panel. The Label Table panel will be an additional table view of the selected label. You can click the Label Table button again to hide to Label Table panel.

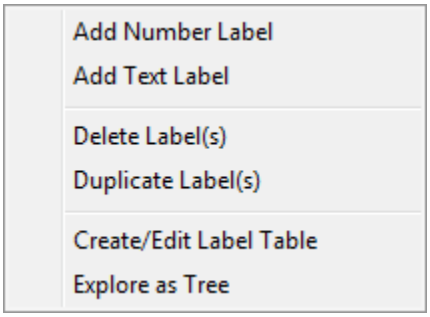
The commands used for editing tables can be used to edit label tables. The following example will set the value of the first cell in the label table "mylabeltable" to 8080 if current is a reference to this object.

settablenum(label(current,"mylabeltable"),1,1,8080);



Popup Menu

You can also edit labels by right clicking on a label. A popup menu will appear, giving you the options below. These options affect the label right-clicked on, not the entire selection.



Add Number Label - This option adds a number label to the object, the same as the Add Number Label button at the bottom.

Add Text Label - This option adds a text label to the object, the same as the Add Text Label button at the bottom.

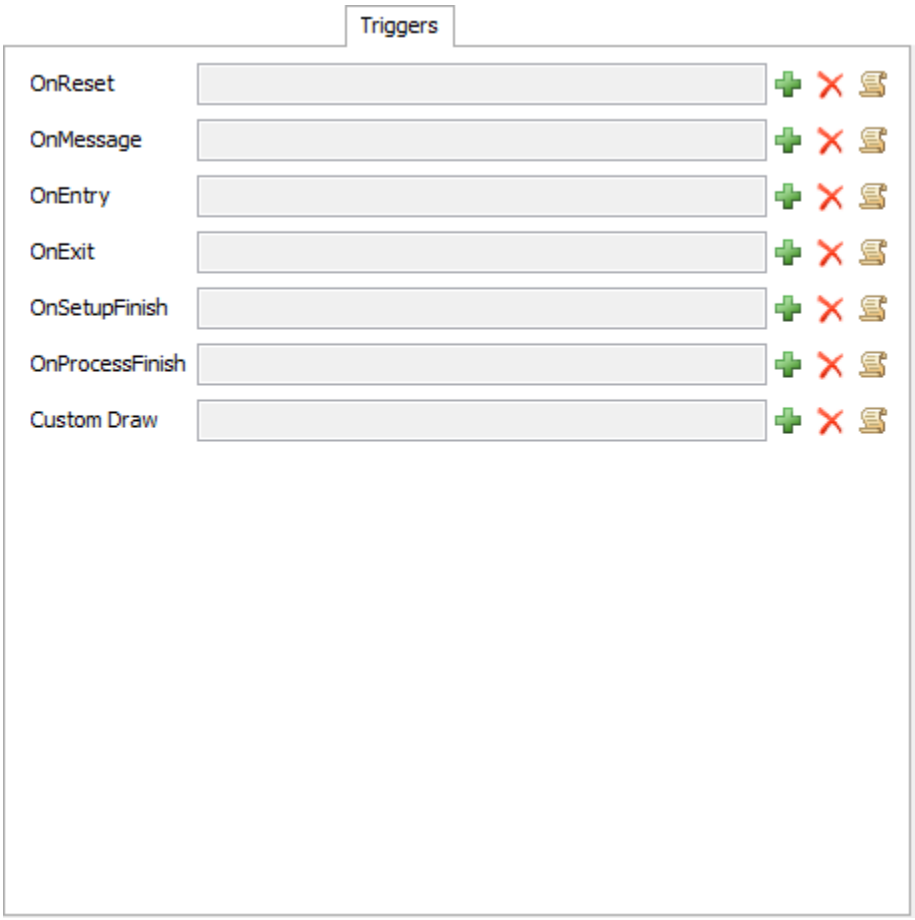
Delete Label(s) - This option deletes the selected labels.

Duplicate Label(s) - This option duplicates the selected labels.

Create/Edit Label Table - This option lets you use a label as a two-dimensional table. It brings up a table edit window to edit the label as a table. To get and set values in this table during the model run, you can use the gettablenum() and settablenum() commands, passing a reference to the label as the first parameter using the label() command. For more information on these commands, refer to the command summary. Example: gettablenum(label(current, "curitemtype"), 4, 5);

Explore as Tree - This option lets you explore the selected label in a tree view.

Triggers Page



Each object may have a different set of triggers. For more information about how triggers and pick lists work, refer to the Picklists page.

OnArrival: This function is called on a NetworkNode when a traveler arrives at the node. If this function returns a non-zero value, then the traveler's next path to go to will be changed to the path specified by the return value. This return value is the rank of the next path, as shown in the NetworkNode tab page. See OnArrival trigger picklist.

OnBreakDown: This function is called on an object when its MTBF expires. See Breakdown/Repair Trigger picklist.

OnContinue: This function is called on a NetworkNode when a traveler continues on to the next path leading out of the node. See OnContinue trigger picklist.

OnConveyEnd: This function is called on a Conveyor when a flowitem reaches its end. See Process Finish Trigger picklist.

OnCreation: This function is called on a Source when it creates a new flowitem. See Creation Trigger picklist.

OnEndCollecting: This function is called on a Queue when it has reached its batch limit. See Process Finish Trigger picklist.

OnEndDwellTime: This function is called on a Rack when a flowitem's dwelltime has expired and it is ready to leave. See Process Finish Trigger picklist.

OnEmpty: This function is called on a fluid object when all of the material that it was holding left and its content became 0. See Empty/Full Trigger picklist.

OnEntry: This function is called on an object when a flowitem is moved into it. See Entry/Exit Trigger picklist.

OnExit: This function is called on an object when a flowitem is moved out of it. See Entry/Exit Trigger picklist.

OnFallThroughHighMark: This function is called on a reservoir when the content falls below the designated high mark. See Rise/Fall Through Mark Triggers picklist.

OnFallThroughLowMark: This function is called on a reservoir when the content falls below the designated low mark. See Rise/Fall Through Mark Triggers picklist.

OnFallThroughMiddleMark: This function is called on a reservoir when the content falls below the designated middle mark. See Rise/Fall Through Mark Triggers picklist.

OnFull: This function is called on a fluid object when its content reaches its maximum content. See Empty/Full Trigger picklist.

OnLoad: This function is called on an Operator or Transport when it completes loading a flowitem. See Load/Unload Trigger picklist.

OnMessage: This function is called on an object when another object sends a message to it using the sendmessage() or senddelayedmessage() commands. See Message Trigger picklist.

OnProcessFinish: This function is called on an object when its process time has expired. See Process Finish Trigger picklist.

OnReceiveTaskSequence: This function is called when the TaskExecuter receives a new TaskSequence. See OnReceiveTaskSequence Triggers picklist.

OnRepair: This function is called on an object when its MTTR expires. See Breakdown/Repair Trigger picklist.

OnResourceAvailable: This function is called when a downstream resource of a Dispatcher becomes available. See OnResourceAvailable Trigger picklist.

OnRiseThroughHighMark: This function is called on a reservoir when the content rises above the designated high mark. See Rise/Fall Through Mark Triggers picklist.

OnRiseThroughLowMark: This function is called on a reservoir when the content rises above the designated high low. See Rise/Fall Through Mark Triggers picklist.

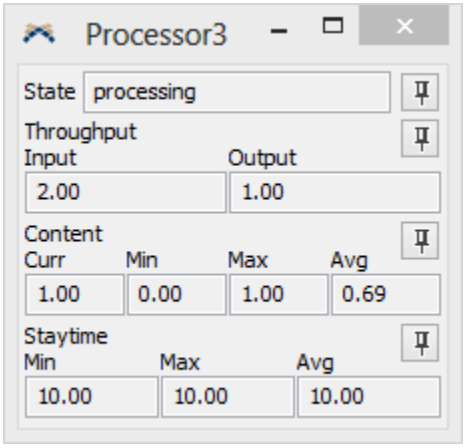
OnRiseThroughMiddleMark: This function is called on a reservoir when the content rises above the designated middle mark. See Rise/Fall Through Mark Triggers picklist.

OnSetupFinish: This function is called on an object when its setup time has expired. See Process Finish Trigger picklist.

OnUnload: This function is called on an Operator or Transport when it completes unloading a flowitem. See Load/Unload Trigger picklist.

Custom Draw Code - This pick list allows you to define your own draw code for the object. If this field returns a 1, then the object's default draw code will not be drawn. Note that an object's draw code is different than its 3D shape being drawn. While most objects just show their 3D shape and don't have any draw code, some objects, like conveyors and racks, need more dynamic drawing capability, rather than a static 3D shape to draw. Returning 1 overrides this special drawing code, not the drawing of the object's 3D shape. To hide the 3D shape, un-check the show 3D shape box in the General tab page.

Statistics Window



The Statistics window is not part of an object's properties window, rather it is a separate window that is accessible from the Quick Properties window under the Statistics Tab. This Statistics tab only displays when an object is selected in the 3D view and the model time is greater than 0.

The statistics shown in this window include the object's current state, throughput, content and staytime. If the object is a TaskExecuter object, it will also display the total distance travelled.

- Pins a statistic to a dashboard. This button creates a new dashboard statistic object and pins it to the selected Dashboard.

Other Properties Pages

1. Container
2. Container Functionality
3. Display
4. NetworkNode
5. NetworkNodes
6. Speeds
7. Traffic Control

Container Page

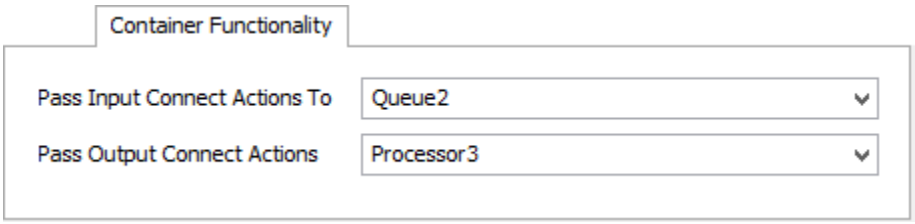
This page will appear in a flowitem that is a Container type. For more information see the Flowitem Bin.

Pack Contents - This drop down will show a list of all available packing methods. Setting a packing method will cause the packing method code to be fired any time a flowitem is placed inside this flowitem.

This Page is Used By

Flowitems

Container Functionality Page



This page will appear in the VisualTool's Properties window if the VisualTool has objects inside of it.

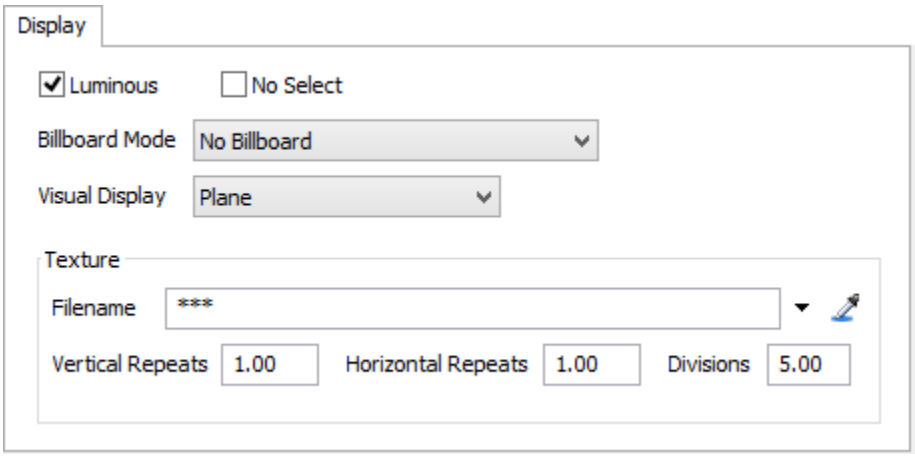
Pass Input Connect Actions To - Set this pull-down list to the object that should be connected to the input port of the Visual Tool. Any objects you subsequently connect to the Visual Tool will automatically connect from the Visual Tool to this object. For more information, refer to the Using the VisualTool as a Container section.

Pass Output Connect Actions - Set this pull-down list to the object that should be connected to the output port of the Visual Tool. Subsequently, when you connect the Visual Tool to another object, it will automatically connect from this object to the Visual Tool. For more information, refer to the Using the VisualTool as a Container section.

This Page is Used By

VisualTool

Display Page



Luminous - If this box is checked, the object will appear to give off its own light and will have gradual shadows caused by the defined light sources.

No select - If this box is checked, the object cannot be selected using the mouse in the Orthographic / Perspective views.

Minimum Magnification - This is the minimum magnification that the object will be visible for.

Maximum Distance - This is the maximum distance that the object will be visible for. If your view is further than this distance the object will not be displayed.

Billboard Mode - This option is usually only used when "Visual Display" is set to Plane. Options include:

- **No Billboard -** The visual tool will display based on its set rotation.
- **Vertical Billboard -** This will cause the plane to stand vertically in the model and always face toward the camera along the same viewing plane.
- **All Around Billboard -** This will cause the plane always present its face to you no matter what the viewing angle is.
- **Screen Locked Billboard -** Locks the Visual Tool in front of the view.

Visual Display - This selects the display type the VisualObject will be. The available types are: Plane, Cube, Column, Sphere, Imported Shape, Text, or Presentation Slide. The various options in this drop-down menu and the subsequent changes to this tab page are described in more detail in the VisualTool section.

Texture

These properties are used to define how a texture is drawn on the 3D object.

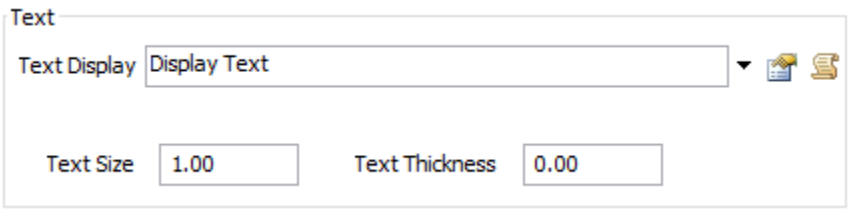
Filename - This file is the bitmap image that will be textured on the object.

Vertical Repeats - This number defines how many times the texture image will be repeated vertically across the image.

Horizontal Repeats - This number defines how many times the texture image will be repeated horizontally across the image.

Divisions - This number is used to define the number of sides on the object if it is a column and the "roundness" of the object if it is a sphere. If the object is a sphere, this number should be relatively high (~20).

Text



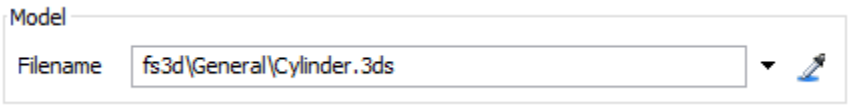
These options are available when Text is selected in Visual Display.

Text Display - This picklist defines what text will be displayed. You may choose from the picklist options, or click the to edit the code directly.

Text Size - This number defines the height of the text in the object. The width of the text will be automatically adjusted to keep the text easily legible.

Text Thickness - This number defines the thickness of the text in the object.

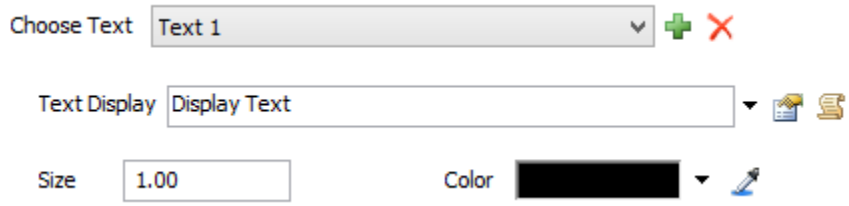
Model



These options are available when Imported Shape is selected in Visual Display.

Filename - Specifies which 3D shape to display. You can use a 3D shape from the model, or import a new shape by clicking the down arrow and selecting Browse...

Presentation Slide



These options are available when Presentation Slide is selected in Visual Display.

Choose Text - Presentation Slides may have multiple text displays. Click the to add a new text object. The options below will enable to specify the text's properties.

Text Display - This picklist defines what text will be displayed. You may choose from the picklist options, or click the to edit the code directly.

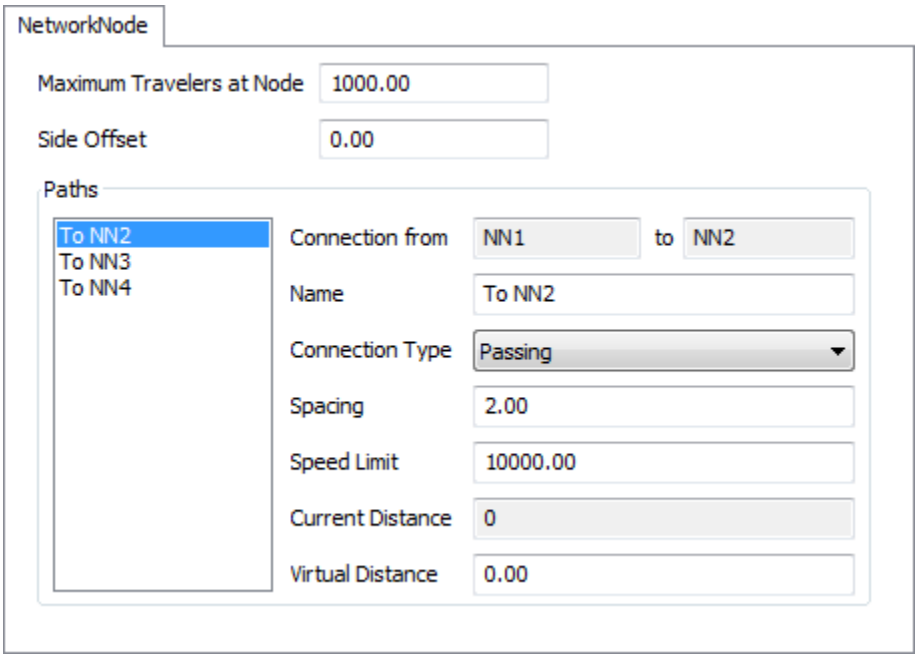
Size - The size of the displayed text.

Color - The color of the displayed text.

This Page is Used By

VisualTool

NetworkNode Page



For more detailed information on these fields, see theNetworkNode documentation.

Maximum Travelers at Node - This number defines how many transporters that are not traveling on the network can be stationed at the node. This would represent the transporters that are not currently executing a travel task, but are doing other things while "stationed" at the node.

Side Offset - This number defines a distance to the right of outgoing paths that travelers will be offset. It does not affect the distance that the traveler travels, but is purely for visual purposes, so travelers going in different directions along the same path don't run over each other.

Paths

Note on NetworkNode connections: Each path between two NetworkNodes contains two one-way connections. This page defines behavior only for the connections extending from this NetworkNode to other NetworkNodes. If you would like to edit behavior for connections extending from other NetworkNodes to this NetworkNode, then open the Properties window of those other nodes.

Name - This field allows the user to name each connection in the network. These names should be descriptive of any special purpose this connection has in the model.

Connection Type - This drop-down list allows the user to define how this connection behaves. There are three options.

No Connection - Transporters cannot travel on this connection. The connection is drawn in red in the view window.

Passing - Transporters are allowed to pass each other on this connection. The connection is drawn in green in the view window.

No Passing - Transporters will not pass each other on this connection. The minimum distance between transporters on the path can be set by the user in the Spacing field of the dialog. These connections are drawn in yellow in the view window.

Spacing - This number determines the minimum distance allowed between two transporters on a connection that is designated as no passing. This is the distance from the back of one traveler to the front of the traveler behind it.

Speed Limit - This number determines the maximum speed that a traveler can travel along this connection.

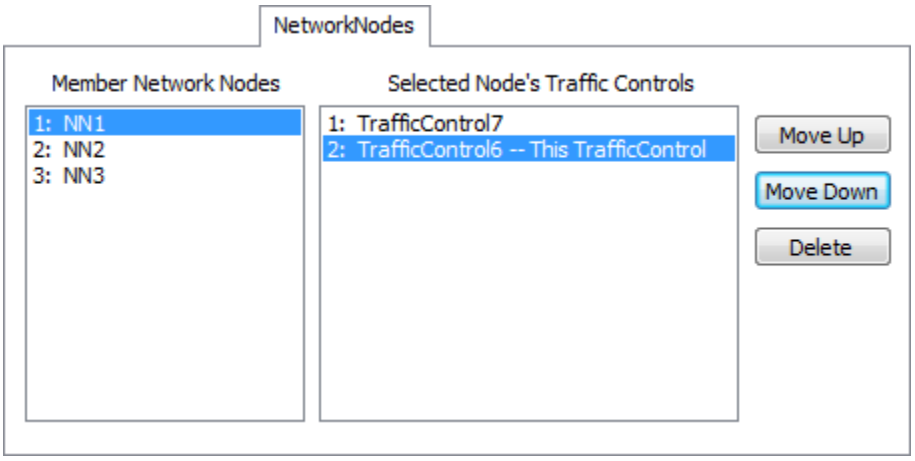
Current Distance - This number shows you the current distance that is being simulated for that connection. If the virtual distance is specified as 0, then it will be the actual distance of the spline path. Otherwise it will be the distance that is specified in the virtual distance field.

Virtual Distance - This number let's you specify an exact distance for the connection.

This Page is Used By

NetworkNode

NetworkNodes Page

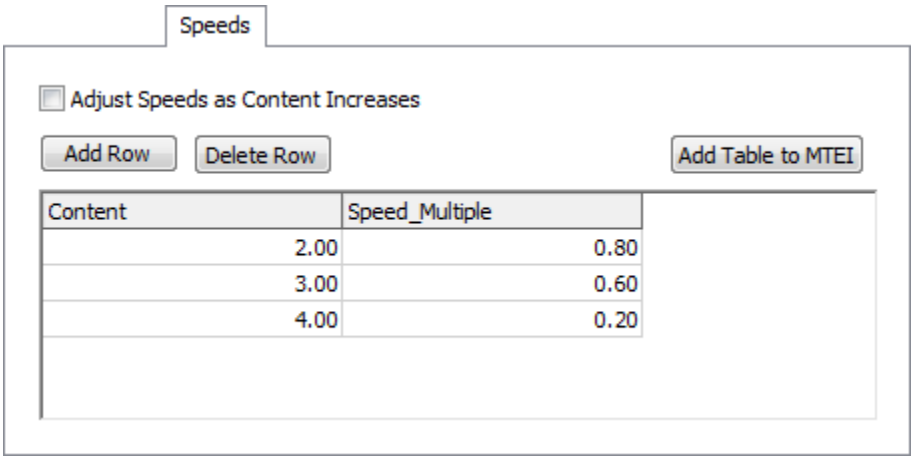


The NetworkNode page of the TrafficControl object lets you edit a TrafficControl's connections to network nodes. The panel on the left shows all NetworkNodes that are connected to the TrafficControl. Select one of these nodes, and the panel on the right will refresh to a new list of traffic controls. The panel on the right shows all traffic controls that are connected to the NetworkNode you have selected on the left. The traffic control you are editing will have an extra " -- This TrafficControl" text, letting you know how this traffic control is situated among other traffic controls. The ranking of traffic controls in the right panel can have a significant effect on the behavior of the network. For more information on traffic control ranking, refer to the section on the TrafficControl. To move a traffic control up in its rank, press the Move Up button. To move a traffic control down rank, press the Move Down button. To delete a NetworkNode's connection to a TrafficControl, press the Delete button.

This Page is Used By

TrafficControl

Speeds Page



Adjust Speeds as Content Increase - If this option is checked, then the table will be used to adjust speeds of travelers within the TrafficControl's area.

Add Row - Click this button to duplicate the selected row. If no row is selected, it will add a row to the end of the table

Delete Row - Click this button to delete the selected row from the table. If no row is selected, it will delete the last row of the table.

Manipulating Speeds with Traffic Controls

You can also use Traffic Controls to modify speeds of travelers as an area becomes more crowded. As the traffic control's content increases, entering travelers will modify their speeds based on the Traffic Control's speed table. For example, if you have entered a row in the table that is a 0.6 speed multiple for a content of 3, then as soon as the content of the traffic control's area becomes 3 or greater, all travelers' max speeds will be decreased to 60% of their normal max speed. Note that the speed change does not take effect until the traveler reaches its next node in the network. If you have an area with multiple traffic controls, then the minimum speed multiple of all of the traffic controls will be applied.

This Page is Used By

TrafficControl

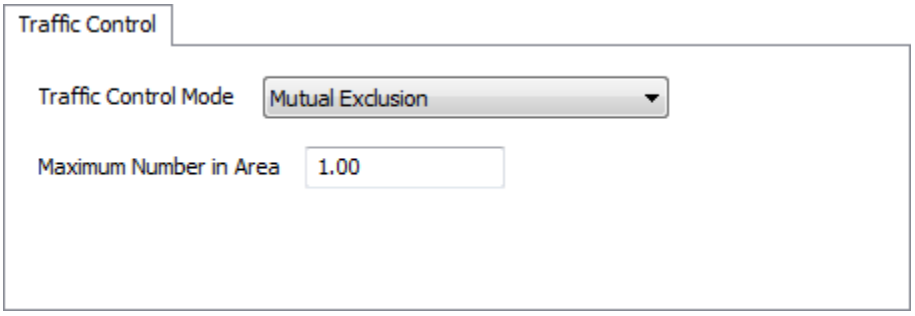
Traffic Control Page

For more detailed information on the TrafficControl object, refer the TrafficControl object.

Traffic Control Mode - This defines the method by which the object controls traffic. There are two options: mutual exclusion or un-timed traffic modes.

Mutual Exclusion

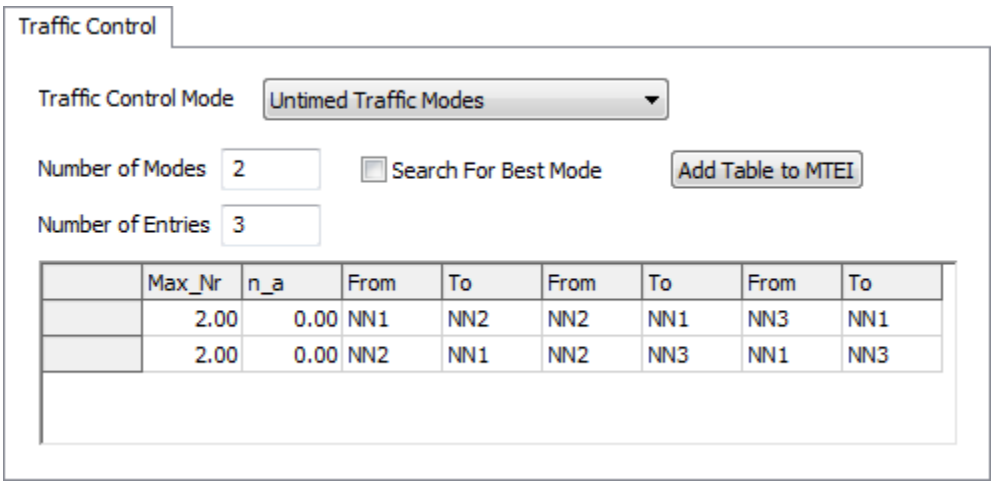
Mutual exclusion is used to only allow a given number of travelers into the Traffic Control's area, regardless of which paths they are on. Here you simply specify the maximum number value.



Maximum Number in Area - This value defines the maximum number of travelers that are allowed to be in the traffic control's area.

Un-timed Traffic Modes

Un-timed traffic modes are used if you want to control traffic based on each individual path in the object's traffic control area. A mode is defined by one row in the modes table. For each mode you can define a set of paths between nodes in the traffic control area. When the traffic control is in a given mode, travelers are only allowed into the traffic control area if the path they are entering on is one of the paths defined in the current mode. The traffic control will stay in the same mode until there are no travelers left in the traffic control area, after which it will take the first traveler that arrives and find a mode that contains that traveler's requested entry path. This is why it is an un-timed mode. There is no limit on the amount of time that a traffic control may stay in the same mode.



Number of Modes - This is the number of modes, or rows in the table. Enter the number of modes you want, and click the Apply button, and the appropriate number of rows will be created.

Number of Entries - This is the maximum number of From/To entries (or columns) that you will need for your modes. If some modes don't need all of the columns, just leave them blank.

Search for Best Mode - If this box is checked, then whenever the traffic control gets an entry request for an entry path that is not in its current mode, it will search through its modes to see if there are any other modes that include all paths already entered as well as the new path. If so, it will change to that new mode

and allow the traveler's entry. Note that this may slow down the model, since the traffic control must search the table every time an entry request is made.

Mode Table Entries

Max_Nr - This value specifies a maximum number of travelers allowed in when the traffic control is in that given mode. It is much like the maximum number value in mutual exclusion mode.

n_a - This value is reserved for future development, when timed traffic modes are implemented.

From/To Entries - For each path of a mode, you specify the node from which the path extends, and the node to which the path extends. Enter the name of the nodes. Note that one entry describes only one direction of a node-to-node connection. Thus to specify both directions of a path, you will need to make two From/To entries, one in each direction.

This Page is Used By

TrafficControl

FlexSim Object Library

1. Overview
2. FixedResources
3. TaskExecuters
4. Travel Networks
5. Visual
6. Fluid Library

FlexSim Object Library Overview

The FlexSim library is made up of objects that are designed to interact with each other in a way that is easy to use and understand. These objects have been implemented using an object-oriented methodology, which involves a super-class/subclass hierarchy. Subclass objects inherit attributes and default behavior from their super-classes while specializing that behavior to fit a specific situation. In FlexSim, most objects in the library have been created as one of two general object types, or super-classes. These two general types we refer to as FixedResources and TaskExecuters.

FixedResources

FixedResources are stationary objects in your model that may represent steps in your process, such as processing stations or storage areas. Flowitems progress through the model by entering, being processed by, and then finishing at these steps in the model. When a flowitem is finished at one step, it is sent on to the next step, or FixedResource, in the model.

TaskExecuters

TaskExecuters are used as shared, mobile resources in the model. They may be operators that are required in order for a given step to process a flowitem, or they may transport flowitems between steps. They can perform many other simulation functions as well.

As you become more experienced in using FlexSim, you will realize that the distinction between FixedResources and TaskExecuters can sometimes become very blurred. TaskExecuters have the capability of simulating FixedResource-like processing steps in a model, while FixedResources can also be configured to travel or operate as shared resources. The only difference is the perspective from which you approach the problem.

Fluid Objects

There are eleven objects that are designed to handle fluid material. Nine of them cannot interact with FlexSim's Discrete objects, but two of them are designed to work as an interface between the Fluid Objects and the Discrete Objects. More information can be found in the Fluid Objects Overview.

Learning Suggestions

In getting to know the FlexSim object library, we suggest that you first read the help section for the FixedResource. Then read the help section for the TaskExecuter, as well as task sequence help. Once you are familiar with how these two general types of objects work, you can learn the specialized functionality for subclasses of these two general types. These subclasses are listed below.

FixedResources

Source
Queue
Processor
Sink
Combiner
Separator
MultiProcessor
Rack
Conveyor
MergeSort
BasicConveyor
BasicFR

TaskExecuters

Dispatcher
TaskExecuter

Operator
Transporter
Elevator
Robot
Crane
ASRSvehicle
BasicTE

Travel Networks

NetworkNode
TrafficControl

Visual

VisualTool

Fluid Objects

Ticker
FluidTank
FluidGenerator
FluidTerminator
FluidMixer
FluidBlender
FluidSplitter
FluidPipe
FluidProcessor
ItemToFluid
FluidToItem

FixedResources

1. Concepts

Fixed Resource Objects

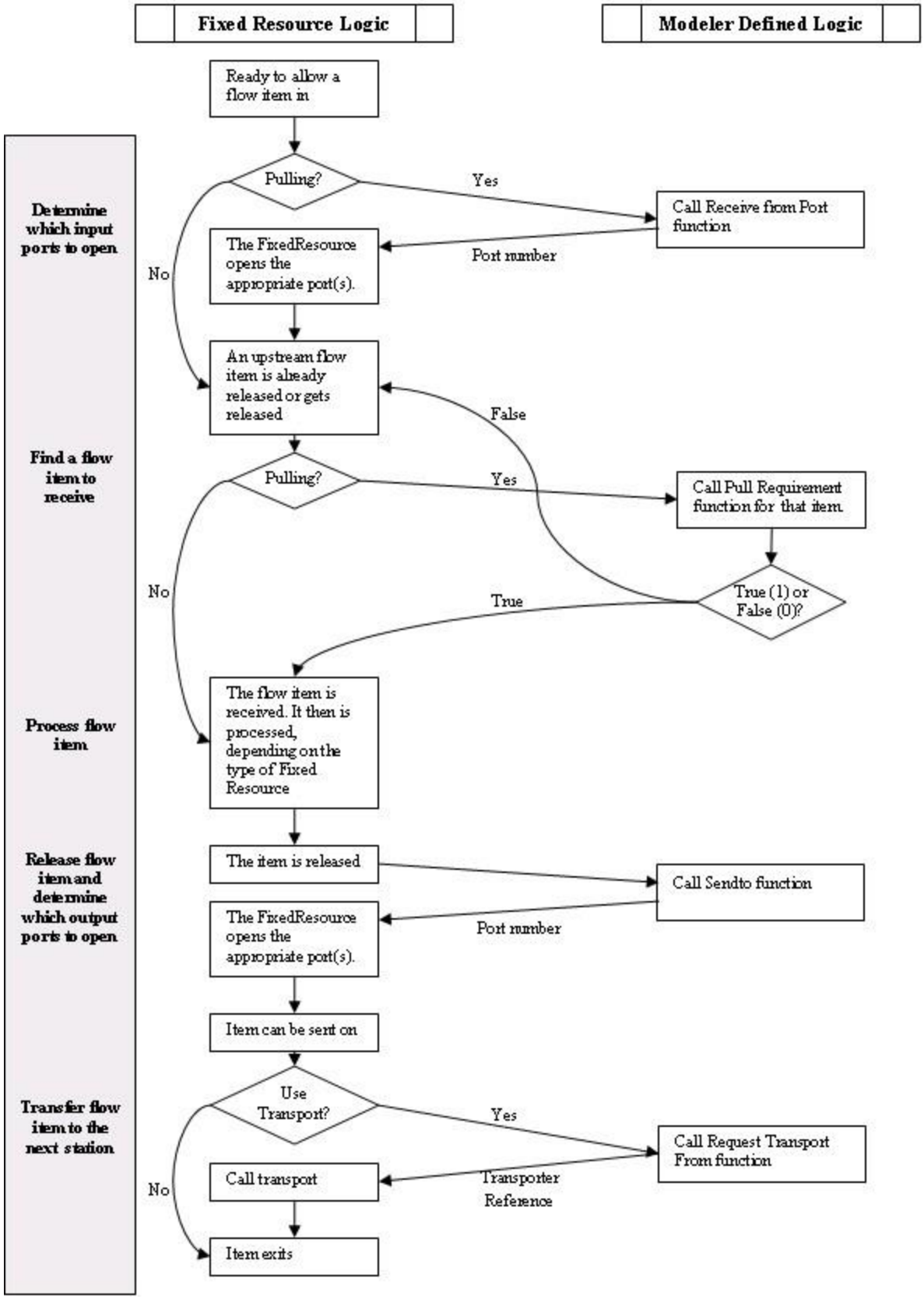
- **BasicConveyor**
- **BasicFR**
- **Combiner**
- **Conveyor**
- **MergeSort**
- **MultiProcessor**
- **Processor**
- **Queue**
- **Rack**
- **Separator**
- **Sink**
- **Source**

FixedResources Concepts

The FixedResource is a superclass of the Source, Queue, Processor, Sink, Combiner, Separator, Rack, Conveyor, and MultiProcessor objects. It defines logic for pulling flowitems into the station, as well as sending the object on. You will not drag this object explicitly into your model, but will instead edit FixedResource logic using the Flow tab of sub-class objects.

Details

The term FixedResource describes a class of objects that handles flowitems in a certain way. They "receive" flowitems through their input ports, do something to those flowitems, then "release" the flowitems to be sent on through their output ports. While different types of FixedResources receive and release flowitems at different times, the processes of receiving and releasing a flowitem are the same for all FixedResources. For example, the Queue can receive several flowitems at the same time. The Queue also releases each flowitem as soon as it enters the Queue. The Processor on the other hand receives exactly one flowitem, processes that flowitem, then releases it and waits until the flowitem has left before receiving the next flowitem. Although the Queue and Processor receive and release flowitems at different times, the processes of receiving and releasing the flowitem are the same for both. Each goes through a certain set of steps for each flowitem that it receives and releases. Some of these steps are automatically handled by the FixedResource, and some allow you as a modeller to define the way flowitems are received and released. All of these modeller-defined inputs can be edited in the Flow tab of an object's Properties window. The following diagram shows the steps that a FixedResource object goes through for each flowitem that it receives and subsequently releases.

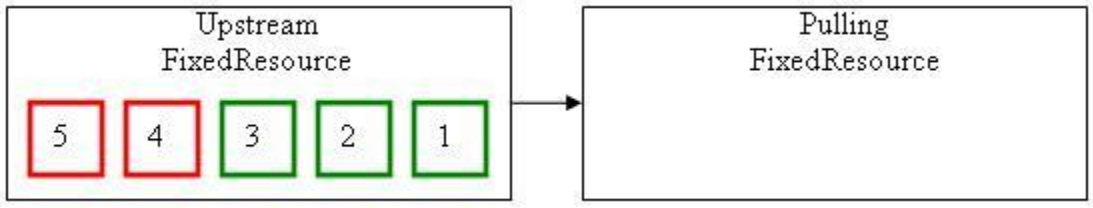

Fixed Resource Logic
Modeler Defined Logic
Ready to allow a flow item in
Pulling?
Yes
Call Receive from Port function
Port number
The FixedResource opens the appropriate port(s).
No
An upstream flow item is already released or gets released
False
Pulling?
Yes
Call Pull Requirement function for that item
True (1) or False (0)?
No
True
The flow item is received. It then is processed, depending on the type of Fixed Resource
The item is released
Call Sendto function
Port number
The FixedResource opens the appropriate port(s).
Item can be sent on
Use Transport?
Yes
Call Request Transport From function
Transporter Reference
Call transport
No
Item exits
Determine which input ports to open
Find a flow item to receive
Process flow item
Release flow item and determine which output ports to open
Transfer flow item to the next station

1. Open input ports and find a flowitem to receive

When the FixedResource becomes ready to receive a flowitem, it checks first to see if it is in Pull mode. If it is in Pull mode, then it calls the Receive from Port function. This function returns a value of the input port number to open. If 0 is returned, then it will open all input ports. When an upstream item is released, it calls the Pull Requirement field for that item. This field should return a true (1) or false (0). If true, then it

receives the flowitem. If false, then it tries calling the Pull Requirement function again for the next flowitem that has been released, or waits until another flowitem is released by an upstream FixedResource. It continues this loop until the Pull Requirement returns a yes (1) for a particular item, upon which it receives that item. If the object is not in Pull mode, however, then the FixedResource skips all of the pulling logic, and simply waits for the first flowitem that becomes available.

Pull Logic Example



The above diagram shows two FixedResources. The Pulling FixedResource is set to pull from the Upstream Fixed Resource. The Upstream FixedResource has released 3 of its flowitems (green), while 2 flowitems are still being processed (red). When the Pulling FixedResource is ready to receive one of the Upstream FixedResource's flowitems, it calls its Pull Requirement function for each of the 3 released flowitems, until the Pull Requirement function returns a yes (1). As soon as a yes is returned, the Pulling FixedResource will receive that item and finish its pulling logic, until it is ready to receive the next flowitem. If all of the 3 calls to Pull Requirement return a no (0), then the Pulling FixedResource will wait. When flowitem 4 is released later in the simulation, the Pulling FixedResource will then call Pull Requirement for item 4, and if it returns yes, item 4 will be received. If it returns no, this process will repeat until a flowitem is found that meets the Pull Requirement.

Note for a FixedResource waiting for an upstream flowitem to be released: When an upstream flowitem is released, the Pulling FixedResource will only call the Pull Requirement on that specific flowitem. It will not call the Pull Requirement on previously released flowitems for which the Pull Requirement has already returned a no (0).

Note on the Send To Port of upstream objects that an object is pulling from: In FlexSim versions before 6: if an object was configured to pull from upstream objects, the Send To Port of those objects was nullified. In version 6, the Send To Port of upstream objects is evaluated in conjunction with the pull strategy. Both the send-to-port and the pull strategy must be true for the item to be pulled.

2. Process the flowitem

Once the flowitem has entered the FixedResource, the item is "processed" according to the type of FixedResource, and then released. For example, if it is a Processor object, then the flowitem will be processed for a certain amount of time. If it is a Queue object, then the product is released immediately. If it is a Conveyor object, then the flowitem will convey down the length of the conveyor and be released when it hits the end.

3. Release the flowitem and determine which output ports to open

When the flowitem is released, the FixedResource calls the Send To Port function. Like the Pull from Port function, this function returns a port number. The FixedResource then opens that port. If the port number is 0, then it will open all of its output ports. Once the ports are opened, the FixedResource waits until a downstream object becomes ready to receive the flowitem. If the FixedResource is configured to reevaluate its Send To Port, then each time a downstream FixedResource becomes available to receive a new flowitem, the upstream FixedResource will reevaluate the Send To Port for that flowitem. It's important to note here that this is only executed when the downstream object becomes available. It does not continuously evaluate just because a downstream object is already available. If you want to manually force a re-evaluation at some other time than when a downstream object becomes available, then you can do so by calling the openoutput() command on the upstream object.

Note on the returned Send To Port value: If the returned port is greater than 0, then that port will be opened. If the returned port is 0, then all ports will be opened. If the returned port is -1, the flowitem will not be released, and should be released explicitly later on using the releaseitem() command, or should be moved out using the moveobject command. When it is released again, the Send To Port function will be called again. A -1 return value is more for advanced users.

4. Transfer the flowitem to the next station

Once the flowitem is released, and a downstream object is ready to receive it, if the "Use Transport" checkbox is not checked, then the item will be passed immediately in to the downstream object. If the "Use Transport" checkbox is checked, then the FixedResource will call the Request Transport from function. This function should return a reference to a TaskExecuter or Dispatcher. If a valid reference is returned, then a default task sequence will automatically be created, and a TaskExecuter will eventually pick the flowitem up and take it to its destination. You can also return a 0 value in the Request Transport From field. If a 0 value is returned, then the FixedResource will assume that a task sequence has been created explicitly by the user, and will not create the default task sequence himself. If you return a zero, then you will need to create the task sequence yourself. You can easily get started at doing this by selecting the "Create task sequence manually" pick option in the Request Transport Field picklist, then manually editing the code from there.

Using a Transport

If an object is configured to use a transport to transport flowitems to downstream objects, then when the downstream object pulls the flowitem or becomes ready to receive the flowitem, instead of immediately moving the flowitem into the next station, the object instead creates a task sequence for a TaskExecuter to travel to the object, pick up the flowitem, travel to the downstream object, and drop it off there. This operation involves several important steps. First, when this happens, the object calls its Request Transport From function, and gets a reference of the object to give the task sequence to. Then the flowitem goes into a "Waiting For Transport" state. This means that the destination for that flowitem has been set in stone, and cannot be changed. Send To Port and pull screening has already finished and decided on sending the flowitem out that port, and this decision will not change. Also, each FixedResource object keeps track of two numbers: the number of flowitems that are in transit to the object, and the number of flowitems that will be transported out of the object, but have not been picked up yet. These two variables are respectively called nroftransportsin and nroftransportsout. Once the object has called the Request Transport From field, it increments its own nroftransportsout variable, and notifies the downstream object, which subsequently increments its own nroftransportsin variable. The object then creates a task sequence of:

1. Travel to the upstream object: Travel task.
2. Load the item: FRLoad task.
3. Break to other task sequences if appropriate: Break task.
4. Travel to the downstream object: Travel task.
5. Unload the item into the downstream object: FRUnload task.

Note that the FixedResource uses frload/frunload tasks instead of regular load/unload tasks. These tasks are just like regular load and unload tasks except that right before the TaskExecuter moves the flowitem, it notifies the FixedResource of the operation being done, so that the FixedResource can appropriately decrement its nroftransportsin/nroftransportsout variable. In summary, the nroftransportsout of the upstream object and the nroftransportsin variable of the downstream object are both incremented at the same time that the Request Transport From function is called and the task sequence is created. The nroftransportsout variable is then decremented just before the TaskExecuter finishes the frload tasks and moves the flowitem out of the upstream object.

The nroftransportsin variable is decremented just before the TaskExecuter finishes an frunload task and moves the flowitem into the downstream object.

The integrity of the nroftransportsin and nroftransportsout variables is essential to the correct operation of the objects involved because each object may screen further input/output based on these variables. For example, consider a queue with capacity of 10. If the queue's current content is 5 and it also has 5 products that have not arrived yet, but are in transit to the queue, then the queue must close its input ports because it may possibly become full, even though at this point it only has 5 products. An incorrect nroftransportsin variable could cause serious problems for the queue's content. What does this mean for you? Two things. First, if an object has chosen to transport a flowitem to a given downstream object, there is no turning back or redirecting the flowitem to a different destination, because this would mess up the proper decrementing of nroftransportsin/out variables. Secondly, be very aware of when you use frload/frunload versus regular load/unload, because this can affect the input/output functionality of the objects. Simply put, each execution of a Request Transport From function should eventually (if not immediately) result in the creation of exactly one frload task to load the item from the upstream object and exactly one frunload task to unload the item to the downstream object. In all other cases, regular load and unload tasks should be used.

TaskExecuters

1. Concepts

TaskExecuter Objects

- **ASRSvehicle**
- **BasicTE**
- **Crane**
- **Dispatcher**
- **Elevator**
- **Operator**
- **Robot**
- **Transporter**

TaskExecuters Concepts

TaskExecuter is the top level class for several objects in the library. Operators, Transporters, ASRSvehicles, Cranes and other mobile resources inherit from the TaskExecuter class. All of these objects can travel, load flowitems, unload flowitems, act as shared resources for processing stations, and perform many other simulation tasks.

To learn more on how to begin using TaskExecuters, refer to Lesson 2 of the Tutorials.

Details

TaskExecuters and their sub-classes are able to execute task sequences, perform collision detection, and execute offset travel.

The TaskExecuter is also a sub-class of the Dispatcher class, and thus a TaskExecuter can actually act as a team leader that dispatches task sequences to other team members. Its handling and dispatching logic have subtle differences from the Dispatcher, however. When the TaskExecuter receives a task sequence, it first checks to see if it already has an active task sequence. If there is no active task sequence, or if the newly received task sequence is preempting and has a priority greater than the currently active task sequence, then it will start executing the new task sequence, preempting the active one if needed. Otherwise it will go through the normal Dispatcher logic. If the task sequence is not passed on immediately, then it will queue up in the TaskExecuter's task sequence queue, and if the task sequence is still in the queue when the TaskExecuter finishes its active task sequence, the TaskExecuter will then execute the task sequence.

User-Defined Properties

All TaskExecuters have the following fields that can be defined by the modeler.

Capacity: This parameter defines a value for the maximum content of the object. In default operation, the object will never load more flowitems than this value specifies.

Note for advanced users on the capacity value: this value can be deceiving if you create your own task sequences. Since the TaskExecuter's first and most important responsibility is to execute task sequences, if you give the object a task sequence to load more flow items than its maximum content, then it will load the flowitems anyway. The only real instance in which the maximum content value is used is for the TASKTYPE_BREAK task. If the TaskExecuter comes to a break task, and it has reached its maximum content, then it will not perform the break, and will continue on its current task sequence instead of breaking to another task sequence. This works in the default case when task sequences are created automatically because each task sequence is responsible for the loading of just one flowitem.

Maximum Speed, Acceleration, Deceleration: These define the TaskExecuter's maximum speed, acceleration, and deceleration. Maximum speed is defined in units of length per unit of time, while acceleration and deceleration are defined in units of length per squared unit of time. If you are defining your model in meters and seconds, for example, the speed value is in m/s, etc. These values are used in defining the object's peak speed and change in speed while executing the task types such as TASKTYPE_TRAVEL, TASKTYPE_TRAVELTOLOC, etc.
Travel Offsets for load/unload tasks: This value determines whether the TaskExecuter should execute offset travel to the load/unload location when it loads or unloads a flowitem. For example, if this is not checked, and the TaskExecuter is traveling on a network, then it will only travel to the network node that is at the load/unload station. It will remain on that node while doing the load.
Rotate while traveling: Here you can specify if you want the object to rotate in the direction that it is traveling. This will have no effect on model output. It is only for visual purposes.
Load Time: This field is executed at the beginning of each load task. Its return value is the delay time that the TaskExecuter will wait before loading the flowitem and moving on to the next task. Note that if the TaskExecuter is configured to travel offsets for load/unload tasks, then it will first travel the correct offset, and then start the load time. Thus the load time is added onto the end of the offset travel time; it is not part of the offset travel time.

Unload Time: This field is executed at the beginning of each unload task. Its return value is the delay time that the TaskExecuter will wait before unloading the flowitem and moving on to the next task. Note that if the TaskExecuter is configured to travel offsets for load/unload tasks, then it will first travel the correct offset, and then start the load time. Thus the load time is added onto the end of the offset travel time; it is not part of the offset travel time.
Break to Requirement: This field is executed when the TaskExecuter comes to a break task or callsubtasks task. The return value is a reference to a task sequence. The logic within this field should search the TaskExecuter's task sequence queue, and find a task sequence that is appropriate to break to.

Offset Travel

Offset travel is a mechanism by which different types of objects can travel differently, yet use the same interface for traveling. For example, an object wants to place an item into a given bay and level of a Rack. The way in which the object travels to the correct location to drop off the item depends on the type of object it is. An operator walks to the bay's location and places the item in the level. A Transporter travels to the bay, but must also lift its fork to the proper height of the level. It can travel in both the x and y direction, but only its fork can travel in the z direction. An ASRSvehicle will only travel along its own x axis, lifting its platform to the height of the level, and then pulling the item from the Rack. Hence, each object implements its travel differently, but the interface is the same: travel to the right spot to place the item in the Rack. Offset travel is essentially the only thing that distinguishes each sub-class of the TaskExecuter. For information on how each sub-class implements offset travel, refer to the "Details" section of an object's help page. Offset travel is used in load and unload tasks, in traveltoloc and travelrelative tasks, and in pickoffset and placeoffset tasks.

The offset travel interface is very simple. Every type of offset request translates into an x,y, and z offset distance, and sometimes a reference to a flow item. For example, if an object is given a traveltoloc task for the location (5,0,0), and its current location is (4,0,0), then it automatically translates that task into an offset request for (1,0,0), meaning it needs to travel one unit in the x direction. A travelrelative task translates directly. For example, a travelrelative task for (5,0,0) tells the object to travel 5 units in the x direction. Load and unload tasks also use offset travel if the "Travel Offsets for Load/Unload Tasks" checkbox is checked. When an object needs to load a flowitem from a station, it queries the station for the location of the item. Also, when it needs to unload an item, it queries the station for the location to unload the item. The station returns an offset x/y/z distance, and the TaskExecuter uses this distance to travel its offset. Also, for a load and unload task, the TaskExecuter has a reference to the item in its offset request. This may or may not affect the way the object travels, depending on the type of object. For example, the Transporter's offset travel mechanism is implemented so that if there is an item, or in other words, if the Transporter is loading or unloading an item, the Transporter will lift its fork in the z direction. If there is no item, or in other words, if the Transporter is doing a traveltoloc or travelrelative task, then it will actually travel in the z direction, instead of lifting its fork.

Offset values should be calculated relative to the x/y center of the object, and the z base of the object. For example, a robot is positioned at location (0,0,0). It has a size of (2,2,1). From this the x/y center and z base can be calculated as the location (1,-1,0) (Note: y size extends along the negative y-axis). All offset calculations should be made from this (1,-1,0) location. While giving the robot a traveltoloc task will automatically make this calculation for you, sometimes it is necessary to calculate this location manually and use a travelrelative task. If the robot is given a travelrelative task of (1,0,0), this means that the correct position to travel to is one unit to the right of the robot's x/y center and z base. This translates to the location (2,-1,0). Note that this does not mean that the robot will travel such that its own location is (2,-1,0). Neither will it travel such that its x/y center and z base are at that location. Because it is a robot, it will rotate and extend its arm so that the end of the arm is at the location (2,-1,0). Its actual location will not change at all. Thus the calculation from the object's x/y center and z base allows you to specify a desired destination location which is the same for all objects, but which allows each type of object to handle that destination location differently.

Collision Detection

The TaskExecuter and its sub-classes have the capability of detecting collisions with other objects. Collision detection is performed by adding collision members to a TaskExecuter, then adding collision spheres to it and its collision members, then executing logic when one of the spheres of the TaskExecuter

collides with one of the spheres of one of its collision members. Each collision sphere you specify has a location in the TaskExecuter's frame of reference, and a radius. The TaskExecuter repetitively checks for collisions at time intervals that you specify. At each collision check, the TaskExecuter checks for collisions on all of its collision spheres with all of the collision spheres of all of its collision members. If a collision is found, then the TaskExecuter fires its collision trigger. It does not fire the collision trigger of the object with whom it is colliding. The other object's collision trigger will fire if and when it does its own collision checks. Note that the collision trigger fires for a specific sphere-to-sphere collision. This means that within one collision check the collision trigger can fire several times, once for each sphere-to-sphere collision encountered in the check.

Be aware that you can very easily cause your model execution speed to decrease significantly if you are not careful with collision detection. For example, if a TaskExecuter has 5 collision spheres and 5 collision members, and each of its collision members has 5 collision spheres, then each collision check will need to check for 125 sphere-to-sphere collisions. If all 6 TaskExecuters are checking for collisions, then 750 sphere-to-sphere checks are being made at each collision check interval of the model. This can slow the model down considerably, especially if your collision interval is very small.

You can turn collision detection on and off for a given TaskExecuter by using the setcollisioncheck() command. See the command summary for more information on this command.

States

The TaskExecuter's states are purely dependent on the types of tasks that the TaskExecuter performs. Many tasks are associated with a hard-coded state, but with some tasks the modeler can specify an explicit state for the TaskExecuter to be in when executing that task. Here are some of the states that you will see often with a TaskExecuter. For more information on tasks and task sequences, refer to Task Sequences.

Travel Empty: The object is traveling to a destination object and doesn't contain any flowitems. This state is exclusively associated with the TASKTYPE_TRAVEL task.

Travel Loaded: The object is traveling to a destination object and has load one or more flowitems. This state is exclusively associated with the TASKTYPE_TRAVEL task.

Offset Travel Empty: The object is performing offset travel and doesn't contain any flowitems.

Offset Travel Loaded: The object is performing offset travel and has loaded one or more flowitems.

Loading: The object is loading a flowitem. This state corresponds to the TASKTYPE_LOAD task, and applies only to the time when the object has finished its offset travel and is waiting its modeler-defined load time before loading the item.

Unloading: The object is unloading a flowitem. This state corresponds to the TASKTYPE_UNLOAD task, and applies only to the time when the object has finished its offset travel and is waiting its modeler-defined unload time before unloading the item.

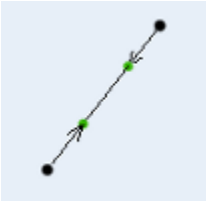
Utilize: The object is being utilized at a station. This state is usually used for an operator, when the operator has arrived at the station and is being utilized for a process, setup, or repair time. The utilize state is usually associated with a TASKTYPE_UTILIZE task, but that task can also specify a different state. Also, other task types, like TASKTYPE_DELAY, can use the utilize state.

Blocked: The object is currently traveling, but is blocked on the network.

Travel Networks

1. NetworkNode
2. TrafficControl

NetworkNode



Topic List

- **Overview**
- **Details**
- **Connecting NetworkNodes to Each Other**
- **Configuring Paths**
- **Configuring Paths Through the Properties Window**
- **Dynamically Close Node Edges**
- **Acceleration/Deceleration on Node Edges**
- **Connecting NetworkNodes to Objects**
- **Connecting NetworkNodes to TaskExecuters**
- **Viewing Connections**
- **Maximum Number of Travelers**
- **Virtual Exits**
- **Commands**
- **Changes to the Distance Table**
- **States**
- **Properties Pages**
- **Related Topics**

Overview

NetworkNodes are used to define a network of paths that transporters and operators follow. The paths can be modified using spline points to add curvature to the path. By default, objects travelling on a network will follow the shortest path between their origin and destination. To learn how to begin using NetworkNodes, refer to Lesson 3 of the FlexSim tutorials.

Details

Connection of travel networks is done in three steps:

1. Connecting NetworkNodes to each other.

2. Connecting NetworkNodes to the objects they will act as gateways to.

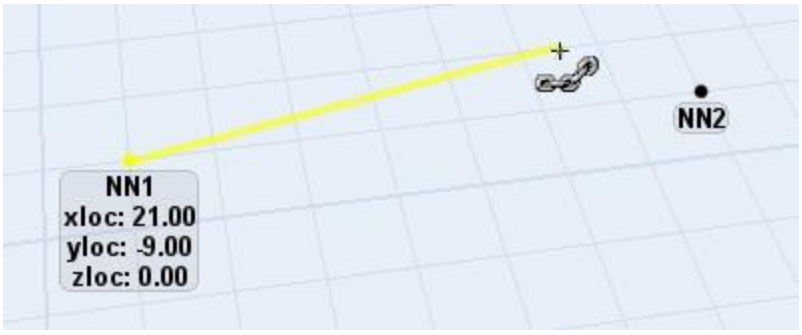
3. Connecting TaskExecuters to the NetworkNodes they will be stationed at for the beginning of the simulation.

Connecting NetworkNodes to Each Other

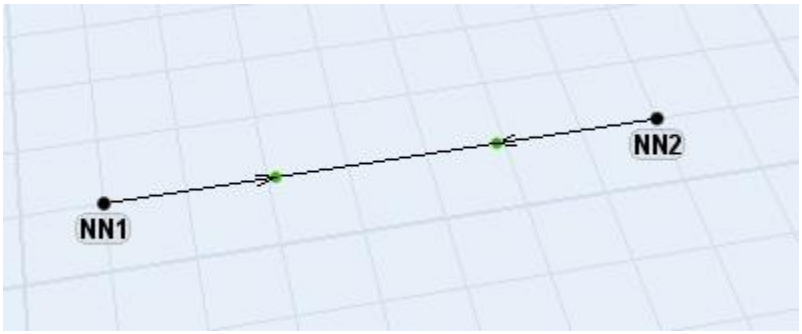
Each NetworkNode can have several paths connecting it to other NetworkNodes. Each path represents two single-direction travel paths. Each path direction can be configured independently.

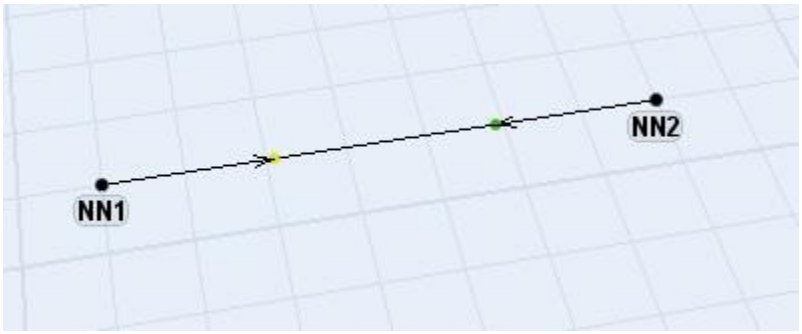
Configuring Paths

To create a path between two network nodes, hold down the 'A' key, click on one network node, and drag to the other node.

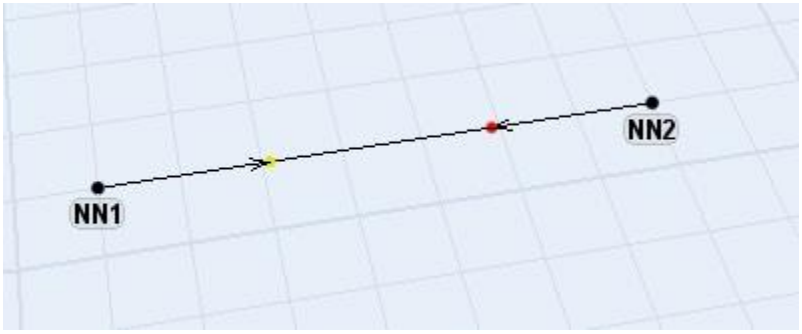


This will create two one-way, passing connections between the nodes. The path is drawn as a green tape between the two nodes. The tape is divided into two sides. Each side describes one direction of the path. Subsequent 'A' drag connections will toggle one direction of the path between passing and non-passing (green and yellow). The direction you toggle is determined by which node you drag from and which node you drag to. The diagram below shows two paths. The first is a passing connection in both directions. The second is a passing connection going right-to-left, and non-passing going left-to-right. The sides of the tape and which direction they describe is determined in the same manner as the American road system: you drive down the right side of the road.



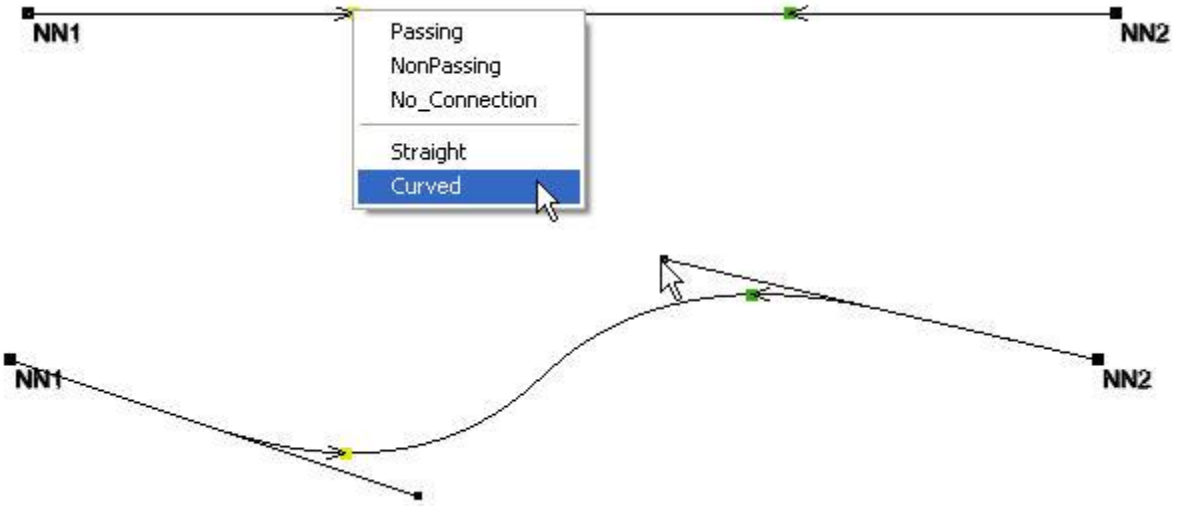


A 'Q' drag connection will toggle one direction of the path as "No Connection", which means travellers aren't allowed to travel in that direction. This type of connection is drawn in red. The diagram below shows a non-passing connection going left-to-right and a no connection going right-to-left. If a 'Q' connection is made in both directions, the whole connection will be deleted.



You can also change the connection type of a given colored box by right clicking on the box and selecting a menu option, or by holding the 'X' key down and clicking on the box.

By default, connections are made as straight paths between nodes. You change these connections to be curved connections by right-clicking on one of the connection's colored boxes and selecting Curved. Two smaller boxes, called spline control points, will be created, and you can move them to create a curved path.



You can also configure how network node connections are made by default through the Travel Networks modeling utility.

Configuring Paths Through the Properties Window

When you open a NetworkNode's Properties window, the Paths page allows you to configure all one-directional paths that extend out of that node. If you want to configure paths going into the node, then edit the properties of the nodes that connect to it. For each path going out of the NetworkNode, you can give it a name, specify the type of travel connection it is, the spacing, the speed limit, and a "virtual" user distance:

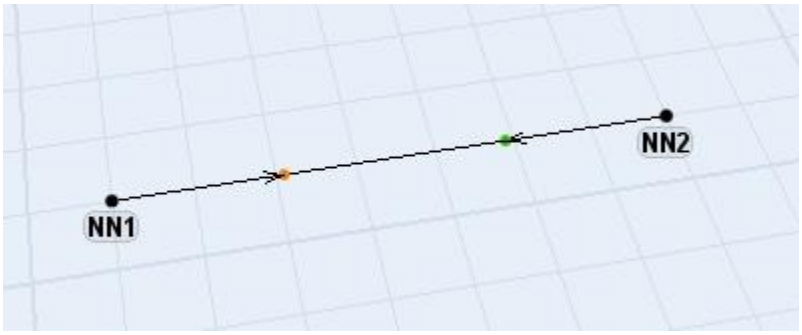
- **Name -** The name of a connection is simply for semantic purposes, and has no effect on model logic.
- **Connection Type -** There are three connection types: No Connection, Passing, and Non-passing. No connection means that no traveller should travel along this path, in the given direction. If this is selected, then the path will be colored red down the corresponding side of the path. Passing means that travelers do not back up along the path, but simply pass each other if speeds vary. Non-passing means that travellers along this path will actually back up, using the spacing value as a buffer distance between them.
- **Spacing -** Spacing only applies if the path is Non-passing. This is the distance to keep between the back of one traveller and the front of another traveller on the path.
- **Speed Limit -** This is a speed limit for the path. Travellers will travel the minimum of their own speed and the speed limit of the path. If the path is a Passing connection, then travellers will accelerate or decelerate to the appropriate speed once the get on the path. If it is non-passing, however, then travellers will immediately change their speed to the appropriate speed, without using acceleration or deceleration.
- **Virtual Distance -** Here you can enter a user-defined distance for the path. This would be used if you want to set a specific distance, and override the path's actual distance, or if it is a very large

distance, but you don't want to have to put the other NetworkNode in a remote part of the model. If 0 is entered, then it will use the actual distance of the path. Otherwise it will use the distance you enter.

Each node's connection has a number associated with it. This is the same as the order of the listing of the connections in the path tab page. The first connection in the list is associated with connection 1, the second with connection 2, and so forth. To get a reference to the NetworkNode that is connected to the node through a given connection number, just use the outobject() command, with the specified connection number.

Dynamically Closing Node Edges

You can dynamically close node paths during the simulation, using the closenodeedge and opennodeedge commands. In these commands, you specify the networknode, and either a number rank or a name of the edge. A closed node edge does not allow any more travellers onto the edge until it has been opened again. However, travellers already on the node edge when it is closed will continue and can exit the edge. A closed node edge is drawn with an orange color, as show below. When the model is reset, all node edges that were previously closed will be opened again.

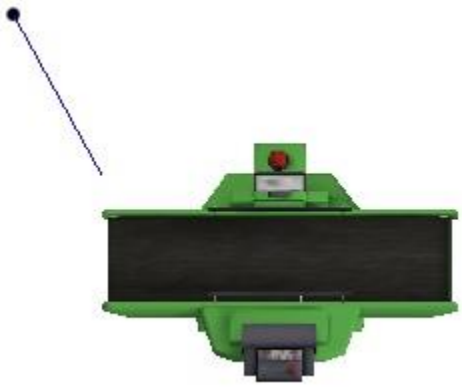


Acceleration/Deceleration on Node Edges

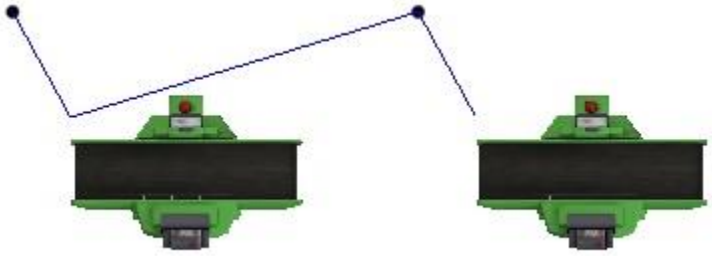
Acceleration and deceleration will work on network edges. Objects will accelerate to their maximum speed as they begin travel on the network. They will also decelerate when approaching the destination. Also, if a traveler is traveling at its maximum speed and it arrives at an edge whose speed limit is less than its maximum speed, then it will decelerate to the speed limit. The deceleration starts once the traveler starts moving along that edge with the lower speed limit, not before it gets there.

Connecting NetworkNodes to Objects

To connect a NetworkNode to some object in the model for which you want the NetworkNode to act as a travel gateway, make an 'A' drag connection between the NetworkNode and the object. The NetworkNode will draw a blue line between it and the top left corner of the object. Making this type of connection means that any TaskExecuter traveling on the network that wants to travel to that object will travel to the NetworkNode that it is connected to.



You can connect several NetworkNodes to the same object. This will cause a TaskExecuter wanting to travel to the object to travel to the NetworkNode nearest to itself that is connected to the object. You can also connect several objects to the same NetworkNode. These capabilities are shown below. The Processor on the left is connected to both the NetworkNode on the left and the NetworkNode on the right. The NetworkNode on the right is also connected to both the Processor on the left as well as the Processor on the right.

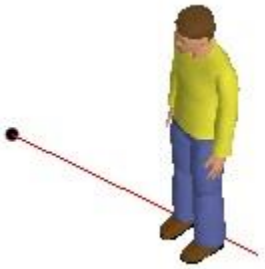


If you connect a node to a station, but don't see the blue line, try moving the NetworkNode around, as the blue line might just be covered up by a grid line.

Connecting NetworkNodes to TaskExecuters

To connect a NetworkNode to a TaskExecuter that you want to travel on the network, make an 'A' drag connection between the NetworkNode and the TaskExecuter. The NetworkNode will draw a red line between it and the top left corner of the object. Making this type of connection means that any TaskExecuter given a Travel task will use the network to get to the destination. It also means that the node that you connect the TaskExecuter to is the first node that it will travel to when it needs to travel through the network. Whenever a TaskExecuter finishes a travel operation, arriving at the node connected to the travel operation's destination object, the TaskExecuter will become inactive at that node, and the red line will be drawn to the TaskExecuter while he is doing other operations in that node's area. What this means is, the next time the TaskExecuter gets a travel task, he must return to the NetworkNode he was inactive at in order to get back onto the network.

(By "inactive" we mean that the TaskExecuter is inactive on the travel network. The object may be actively executing other operations like load, unload, or utilize tasks, but it is not currently traveling on the network.)



You can connect several TaskExecuters to the same NetworkNode. All TaskExecuters connected to a NetworkNode will reset their position to the position of the NetworkNode they were originally assigned to when the model is reset.

If you connect a node to a TaskExecuter, but don't see the red line, try moving the NetworkNode around, as the red line might just be covered up by a grid line.

If you want to connect/disconnect a NetworkNode as a travel gateway to a TaskExecuter, use the 'D' and 'E' keys to connect and disconnect. Connecting in this manner, will cause a blue line to be drawn to the TaskExecuter indicating that other TaskExecuters traveling to that TaskExecuter will travel to the NetworkNode connected to it with the blue line.

Viewing Connections

Once you have built a travel network, you can configure which types of connections you want to be drawn in the Ortho/Perspective view. The network has a set of drawing modes, from showing the most information to showing the least information. These modes are listed below.

Mode 1: Show nodes, paths, object/TaskExecuter connections, spline control points
Mode 2: Show nodes, paths, object/TaskExecuter connections
Mode 3: Show nodes, paths
Mode 4: Show nodes
Mode 5: Show only one node

When you hold the 'X' key down and repeatedly click on a NetworkNode, the whole network will toggle through these modes, showing less information with each 'X' click. When you hold the 'B' key down and repeatedly click on a NetworkNode, the whole network will toggle backwards through the list of modes. You can also select a set of NetworkNodes (hold Ctrl key down and click on several nodes) and then 'X' click on one of that set to have the draw mode toggling only apply to the NetworkNodes you've selected. If you have selected a set of NetworkNodes and 'X' click on a NetworkNode that is not selected, then draw mode toggling will only apply to NetworkNodes that are not selected. This can be very useful if you have a very large model and don't need all spline connections to be drawn.

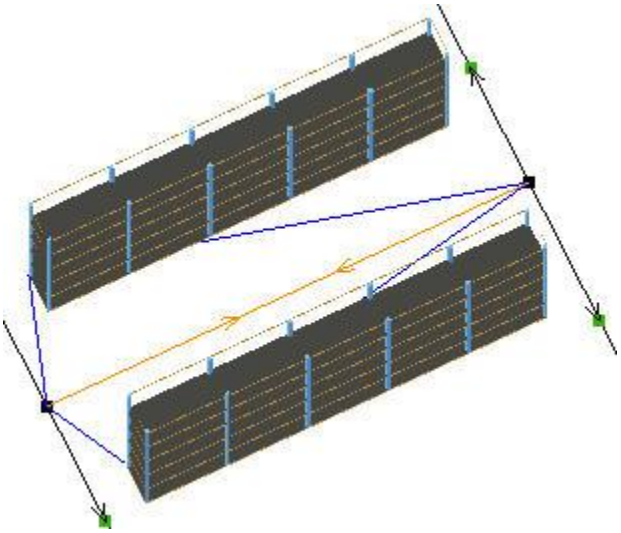
Maximum Number of Travelers

You can specify the maximum number of travelers allowed to be inactive, or stationary, at the node. An inactive traveler is one that is connected to the network node, and is not performing a Travel task, but rather is doing another task or is idle. You can tell a traveler is inactive by the fact that there is a red line drawn between the traveler and the network node.

If the maximum number of stationary travelers for a network node is set to 1, and a traveler is already stationary at the node, then when another traveler arrives at the node it will have to wait until the first traveler leaves the node before it can finish its travel task. Note that this only applies when the second traveler's destination is that node. If the traveler just wants to pass through the node to another node, then it will not have to wait.

Virtual Exits

NetworkNodes can also have virtual exits. Above it was mentioned that when a TaskExecuter finishes a travel task, it becomes inactive at the destination NetworkNode. Once it gets another travel task, it must go back to the original NetworkNode it was at in order to get back onto the network. Virtual exits allow you to specify alternative nodes that the TaskExecuter can travel to in getting back onto the network. Virtual exits are created between NetworkNodes. They are created by holding down the 'D' key and dragging from one NetworkNode to another. An example is shown below.



The above figure shows two Rack objects and two NetworkNodes. The NetworkNodes are travel gateways to the Rack objects (blue lines are drawn between the Racks and the nodes). A two-way virtual exit

connection has also been made between the two NetworkNodes (orange arrows pointing to either node). This means that if a TaskExecuter arrives at the racks through one of the NetworkNodes, and then needs to get back onto the network, it can "exit" the area through any one of the two, whichever has a shorter total distance. Orange arrows pointing out of a given NetworkNode mean that if a TaskExecuter is inactive at that node, it can exit through any one of the NetworkNodes that node is connected to. If it needs to exit through a different node than it entered, it uses the reassignnetnode() command to reassign itself to the new node. Then it simply exits through that node.

To delete a virtual exit, hold the 'E' key down and drag between NetworkNodes in the same direction of the virtual exit connection you want deleted.

Commands

There are several commands you can use to dynamically manipulate networks and transports. These are as follows. More detailed information can be found in the command summary.

reassignnetnode(object transport, object newnode)
This dynamically changes the NetworkNode at which a TaskExecuter is stationed.

redirectnetworktraveler(object transport, object destination)
If a traveler is on a network traveling to a given destination, and you would like it to change its destination in the middle of the travel operation, you can use this command.

distancetotravel(object traveler, object destination)
This command can be used to calculate the distance from a TaskExecuter's current static node to the destination object.

getedgedist(object netnode, num edgenum)
This returns the distance of an edge of the NetworkNode.

getedgespeedlimit(object netnode, num edgenum)
This returns the speed limit of an edge of the NetworkNode.

Changes to the Distance Table

The distance/routing table of all network nodes in the model is kept on a global object in the model called "defaultnetworknavigator". The re-calculation is optimized to only be executed if changes have been made to the network. If you have clicked on a NetworkNode in the model, or if you've 'A' or 'Q' dragged between two NetworkNodes in the model, then the next time you reset the model, the distance/routing table will be re-calculated.

States

The NetworkNode does not implement any states.

Properties Pages

NetworkNode
Triggers
Labels
General

Related Topics

Tutorial: Lesson 3
TrafficControl

TrafficControl

Topic List

- **Overview**
- **Details**
- **Mutual Exclusion**
- **Un-Timed Traffic Modes**
- **Dynamically Changing Modes**
- **Using Several Traffic Controls**
- **Interaction**
- **Resetting a Model**
- **Manipulating Speeds**
- **Customizing Area Entry Rules**
- **States**
- **Properties Pages**

Overview

The TrafficControl is used to control traffic in a given area of a travel network. You build a traffic controlled area by connecting NetworkNodes to the traffic control object. These NetworkNodes then become members of the traffic controlled area. Any path between two nodes that are both members of the same traffic control object is designated as a traffic controlled path, and travelers are only allowed onto that path if given permission by the traffic control object. The traffic control object can be in a mutual exclusion mode, where it only lets a certain number of travelers into the area at any given time, or it can use un-timed traffic modes to only allow travelers onto certain path sections at once.

Details

To connect a NetworkNode to a traffic control, hold down the 'A' key and drag from the traffic control to the node. This will draw a red line between the node and the traffic control. If two NetworkNodes have a path between them, and both NetworkNodes are members of the same traffic control object, then that path is designated as a traffic controlled path, or a member path.

All travelers entering a traffic controlled area must get permission from the traffic control. A traffic control's area consists of all traffic controlled paths as well as the NetworkNode members themselves. What this means is, "entering" a traffic control area is defined as either entering a path that is a member of the traffic controlled area, or arriving at a final destination whose NetworkNode is a member of the traffic control area. However, a traveler is not considered entering the area if it is passing over a NetworkNode that is a member of the traffic controlled area and continuing on a path that is not a member of the traffic controlled

area. Travelers will not need permission in this case. A traveler "exits" a traffic controlled area in one of two ways. Either the traveler is coming from a member path to a path that is not a member of the traffic controlled area, or the traveler is coming from an "inactive" state at a member NetworkNode and continuing on to a path that is not a member of the traffic controlled area. Whenever a traveler exits a full area, room is created for other travelers to enter the area. You can also have an inactive traveler exit a traffic controlled area by calling reassignnetnode(), assigning the traveler to a NetworkNode that is not a member of the area. The table below shows the entry/exit definitions.

Traffic Control Area Entry/Exit Criteria		
	Coming From	**Continuing To**
Area Entry	Non-member path	1. Member path OR 2. Member final destination node
Area Exit	1. Member path OR 2. Inactive state at member node	Non-member path

The traffic control object screens entries into its area using one of two modes: mutual exclusion r un-timed traffic modes.

Mutual Exclusion

When the traffic control is in mutual exclusion mode, it will only allow a certain number of travelers into its area at any given time. Once the area is full, travelers requesting to enter must wait at the edge of the traffic controlled area until another traveler leaves the area and frees up room.

Un-Timed Traffic Modes

When the traffic control is using un-timed traffic modes, it screens entries into the traffic control area based on its table of modes and the entry path of travelers requesting to enter the area. Each row in the mode table represents one mode. Each mode includes a set of entry paths that the traffic control will allow entry for.

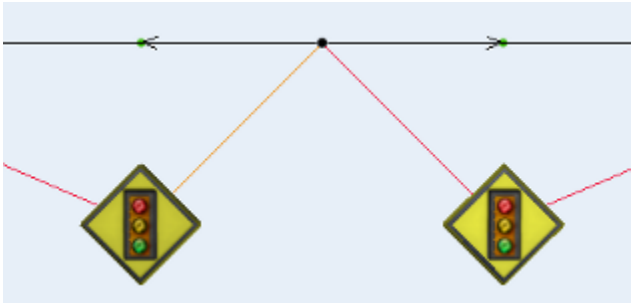
The traffic control selects modes based on entry requests. If there are no travelers in the area, then the traffic control simply waits. When the first traveler requests to enter a certain member path of the area, the traffic control searches its table to find a mode that includes that path. If found, the traffic control goes into that mode, allowing the traveler into the area. Once the traffic control is in a certain mode, it will stay in that mode until all travelers have exited the area. Then it will wait for the next "first traveler" request, and will repeat the cycle.

Dynamically Changing Modes

If the traffic control is set to search for the best mode, then it may change modes dynamically without emptying the area first. The traffic control keeps a record of which paths in the current mode have been traveled on. When a traveler enters on a path, the traffic control marks that path as "dirty" or "traveled on" so to say. Even if the traveler leaves the area later on, the dirty record remains. The record is only reset when the traffic control area is totally emptied. When a traveler requests to enter on a path that is not a member of the current mode, the traffic control searches the rest of the table to see if there is another mode that includes all paths that are currently dirty as well as the path that the traveler is requesting. If it finds one then it changes to that mode and allows the traveler into the area. Otherwise the traveler must wait.

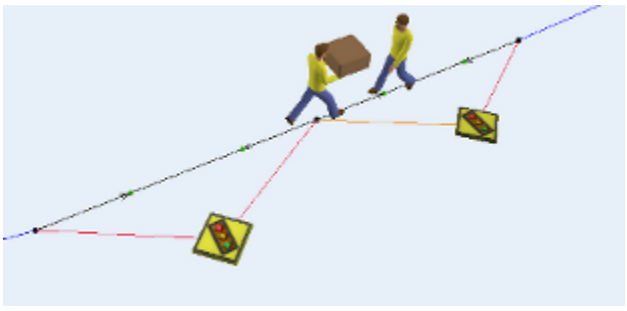
Using Several Traffic Controls

Each NetworkNode can be connected to up to 50 traffic control objects simultaneously. The figure below shows a network node connected to two traffic controls.



Notice that the line drawn from the node to the traffic control on the left is orange, whereas the line to the traffic control on the right is red. These colors show the ordering of the traffic controls in the node's member list. The color ordering is done in ROYGBIV fashion (red, orange, yellow, green, blue, indigo, violet). The first traffic control for that network node has a red line drawn to it, the second an orange line, and so forth. This ordering is very important to the functionality of the model, as well as in avoiding gridlock. When a traveler comes to a network node where it must enter more than one traffic control area, it will request entry into the traffic control areas in the order that those areas are listed on the node. It will only request one traffic control at a time. Once a traffic control gives permission to enter, the traveler has technically entered that traffic control area, even though it may still require permission for entry into other traffic control areas. When transferring between two paths, a traveler will first enter all traffic control areas corresponding to the new path before exiting traffic control areas corresponding to the old path.

Note on gridlock when using several traffic controls: Although using several traffic controls in your model can increase the flexibility and capability of your travel networks, it can also very easily cause gridlock. Traffic control gridlock is usually caused by circular wait. Circular wait happens when one traveler is in a traffic control area that is full, waiting to enter another traffic control area, but the other traffic control area is also full and waiting for a second traveler to exit, but the second traveler cannot leave because it must first enter the area that the first traveler is in. This is a simple example of circular wait. Circular wait can span several travelers and traffic controls and is very difficult to debug. Hence, be very careful when using several traffic controls. Experience has shown that it is beneficial to hierarchically order traffic controls, with some traffic controls in charge of large areas, and some traffic controls in charge of smaller areas. Partial intersection of traffic control areas is strongly discouraged. Either a traffic control should be completely contained within a larger traffic control area, or should be the only traffic control for its area. Travelers should request entry into larger areas before requesting entry into smaller areas. Still, even following these guidelines exactly does not guarantee that a model will not experience gridlock. Much of this type of model building is simply trial and error. The figure below shows a very simple model that still causes gridlock. Notice that the green traffic control on the left is full by the fact that the operator holding the flowitem is in its area, waiting to get permission to enter the blue traffic control's area on the right. But the blue traffic control's area is also full because the operator with no flowitem is in the area waiting to get into the green traffic control area on the left.



Interaction

You can do several things to edit traffic control objects in the orthographic/perspective views. Hold down the 'X' button, and repeatedly click on a traffic control object, and all traffic control objects in the model will toggle between hiding their node connections and showing them. If you only want to hide one traffic control's connections, select the traffic control using the shift key, and then 'X' click on the object and it will hide its connections. While running your model, you can hold the 'V' key down and click on the object, holding the mouse button down. Holding the 'V' key down operates the same as described in the keyboard interaction section. This will draw a line to all travelers that are currently requesting entry to the traffic control area, but have not been given permission yet. If the traffic control's color is not white, then it will draw these lines in its color, allowing you to better distinguish between different traffic control area entry requests.

Resetting a Model

Currently traffic controls are not configured to correctly handle travelers that are placed within a traffic control area when the model is reset. You will need to have all travelers reset to a NetworkNode that is not a member of any traffic control areas. Usually this means adding a NetworkNode off to the side of your model, 'A' connecting all TaskExecuters to that NetworkNode, and then having them enter the model at the beginning of each simulation run.

Manipulating Speeds

You can also use Traffic Controls to modify speeds of travelers as an area becomes more crowded. As the traffic control's content increases, entering travelers will modify their speeds based on the Traffic Control's speed table. For example, if you have entered a row in the table that is a 0.6 speed multiple for a content of 3, then as soon as the content of the traffic control's area becomes 3 or greater, all travelers' max speeds will be decreased to 60% of their normal max speed. Note that the speed change does not take effect until the traveler reaches its next node in the network. If you have an area with multiple traffic controls, then the minimum speed multiple of all of the traffic controls will be applied.

Customizing Area Entry Rules

You can also customize the rules by which a traffic control allows/disallows traveler entry into the traffic control area. For more information on this, refer to the command documentation for trafficcontrolinfo().

States

The traffic control does not implement states at this time.

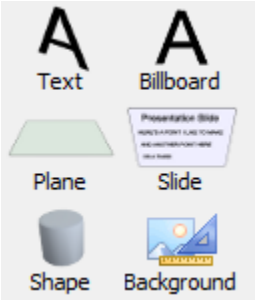
Properties Pages

Traffic Control
Speeds
NetworkNodes
Triggers
Labels
General

VisualTool

1. Overview
2. Example

VisualTool Overview



VisualTools are used to decorate the model space with props, scenery, text, and presentation slides in order to give the model a more realistic appearance. They can be something as simple as a colored box, background, or something as elaborate as an imported 3D graphic model or presentation slide. Another use of the VisualTool is as a container for other objects in your model. When used as a container, the VisualTool becomes a handy tool for hierarchically organizing your model. The container can also be saved in a User Library acting as a basic building block in the development of future models.

Details

Since the VisualTool works differently than other FlexSim objects, a detailed description of how it is used will now be explained. There are many ways you can use the VisualTool object in your model.

- As a container or submodel
- As a plane, cube, column or sphere
- As an imported shape
- As text
- As a presentation slide

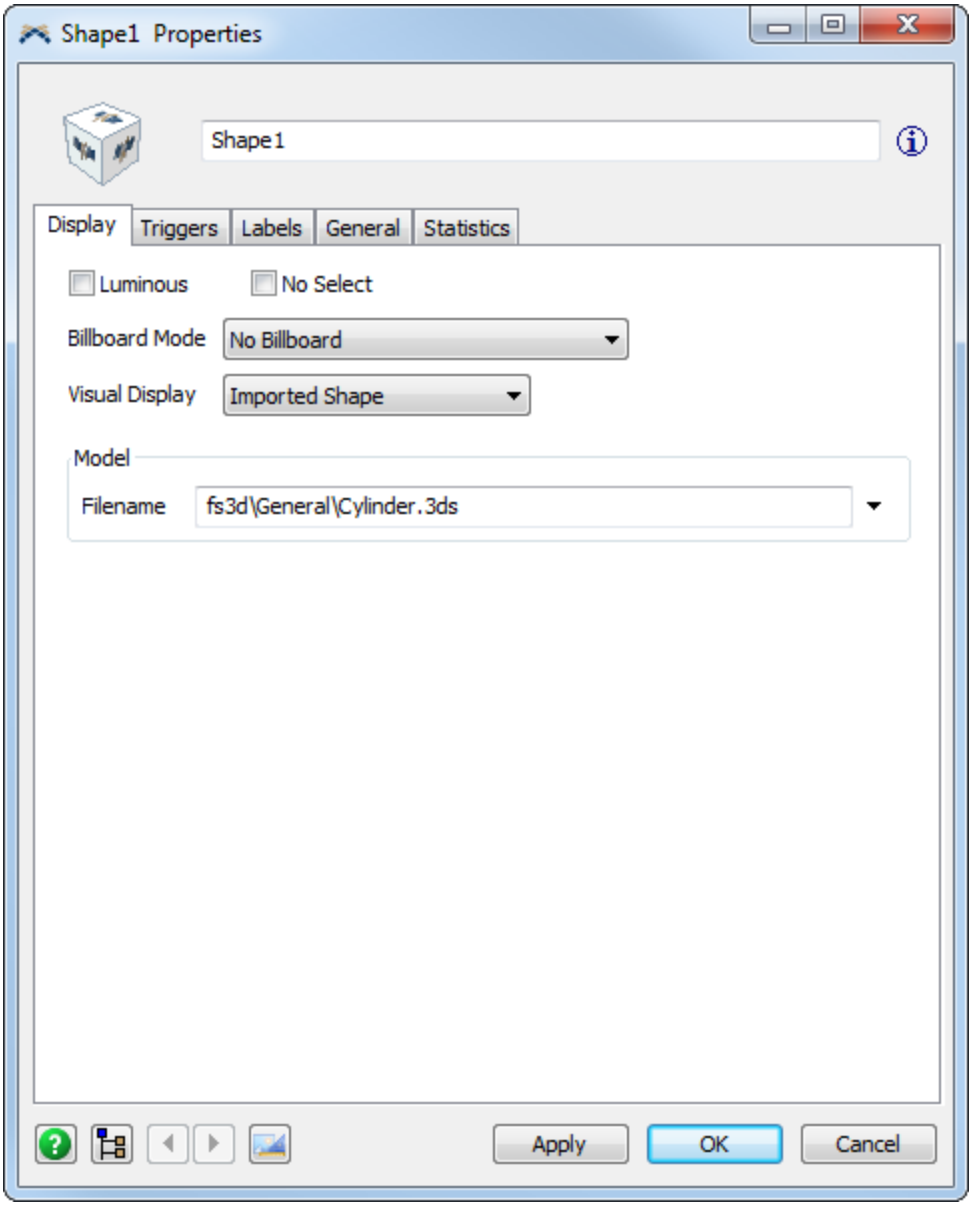
Using the VisualTool as a Container

The default setting for the VisualTool is a plane. When placed in the model the VisualTool will display a gray plane. The size and location of the plane can be set graphically in the Orthographic view window, or by using the VisualTool's Display tab (the use of the Display tab is explained in the section Using the Visualtool as a Plane, Cube, Column, or Sphere). When using the VisualTool as a container it is recommended that you use the default view (a Plane) setting when you start. You can change the visual representation at a future time. For this example we will build a container that has 1 queue, and 2 processors. Flowitems will be passed into the container from a Source that is located outside. The processors will send flowitems to a sink located outside the container.

Statistics on the container can be viewed through the Statistics tab in the VisualTool's properties dialog.

Using the VisualTool as a Plane, Cube, Column, or Sphere

To use the VisualTool as a visual prop in you model is a simple process. On the "Display" tab, choose the type of prop you would like and define the properties.



Plane

A Plane can be defined as a background such as an Autocad layout, a texture or picture, or a patch of color for a specific section of your model. The Plane is the default view for the visual tool. You simply set the size of the Plane and then choose the Texture. The Texture can be repeated in both the vertical or horizontal direction.

Cube, Column, or Sphere

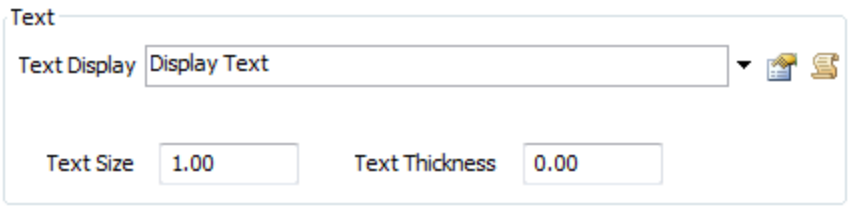
The cube, column, or spheres are simple shapes that can be assigned a texture much like the plane.

Using the VisualTool as an Imported Shape

When using the VisualTool for an imported shape you need to have a 3D model or object that you wish to bring into the model. FlexSim supports a variety of 3D shape formats such as 3D Studio Max (.3ds), Google SketchUp (.skp), VRML 1.0 (.wrl), AutoCAD DXF (.dxf), and Stereo Lithography (.stl).

Using the VisualTool as Visual Text

3D visual text can be added to you model to display labels, statistics, or other information in your model. When the visual display is set to "Text" you are presented with a picklist of options for setting the text.

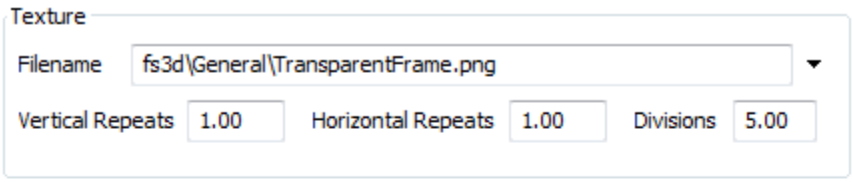


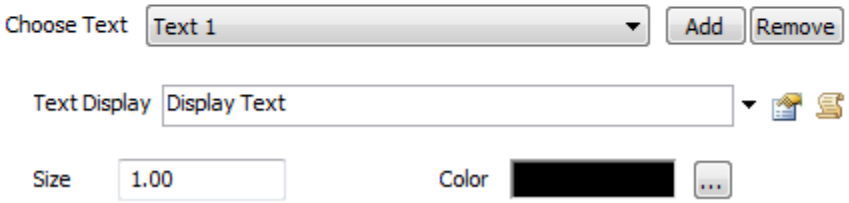
Pick options include simulation time, content, state, outputs, inputs, and many others. If any statistics are selected in the picklist the user must connect the VisualTool center port to the object that you want to display the information about. The text can be edited by selecting the code template button .

Using the VisualTool as a Presentation Slide

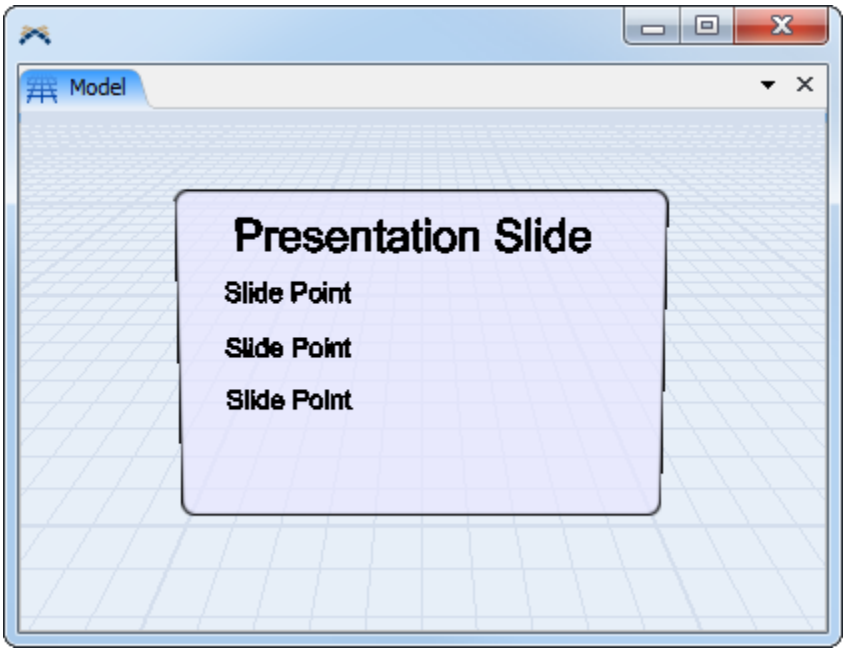
The VisualTool may also be used as a presentation slide much like you would use a slide to develop PowerPoint presentations. Presentation slides are placed in your model to present data, model findings, and presentation points. You can develop a "FlyTo" sequence to view the slides by using the Presentation Builder found in the menu option Tools > Presentation > Presentation Builder.

When the visual display is set to "Presentation Slide" additional options will appear on the Display tab.





Press the Add button to add a new line of text to the slide. The first line of text is labeled "Text 1" and is the slide header. Additional text is added to the slide as line items. For example, if you were to add four text items to the presentation slide you would see the following:



Each text is given the default location on the slide as shown. By selecting the desired text in the drop down box, you can also change the text that is displayed using the Text Display list, the size of the text and the text color.

You can apply any background by selecting a texture on the Display tab or change the background color by selecting the color on the General tab.

Properties pages

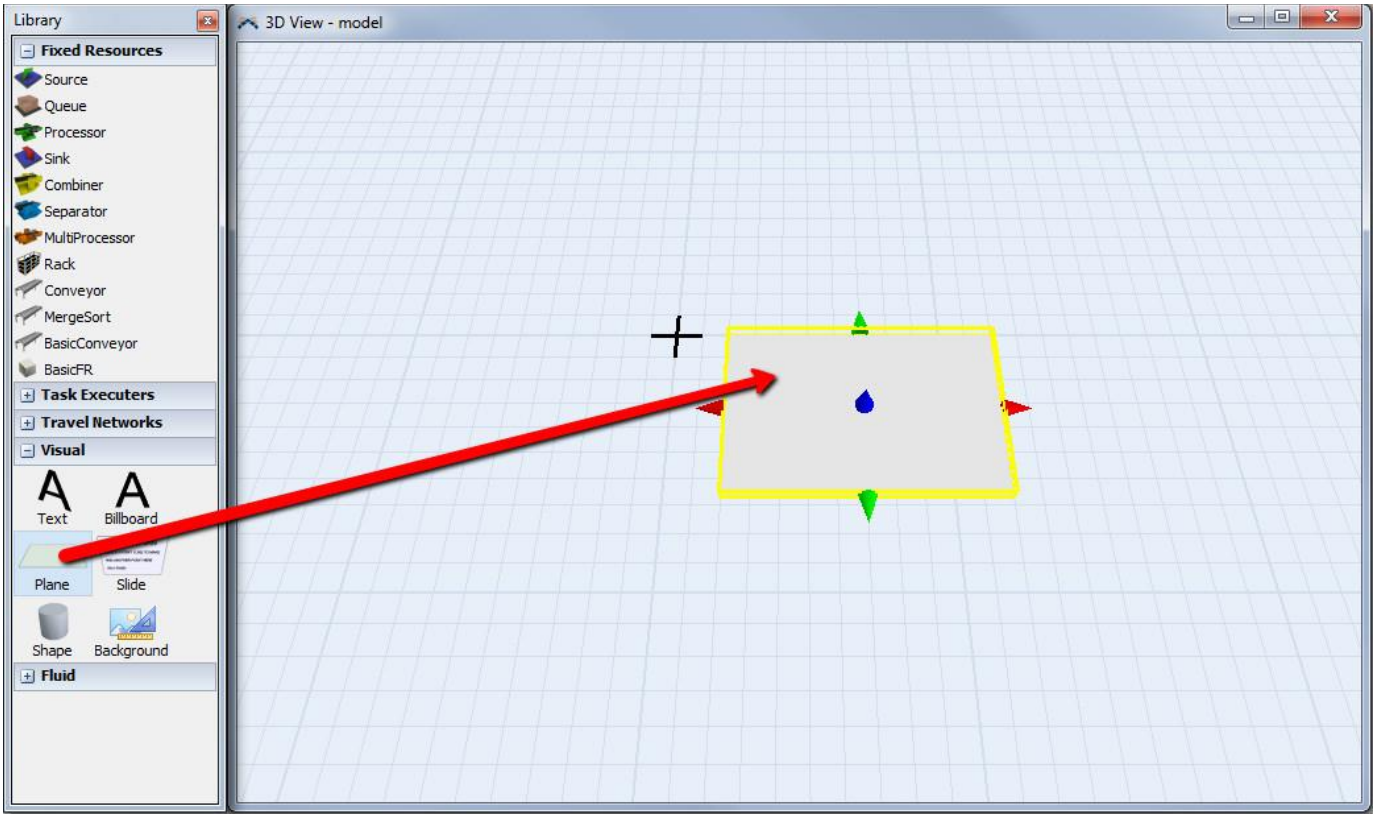
Display
Container Functionality
Triggers
Labels
General

VisualTool Example

Using the VisualTool as a Container

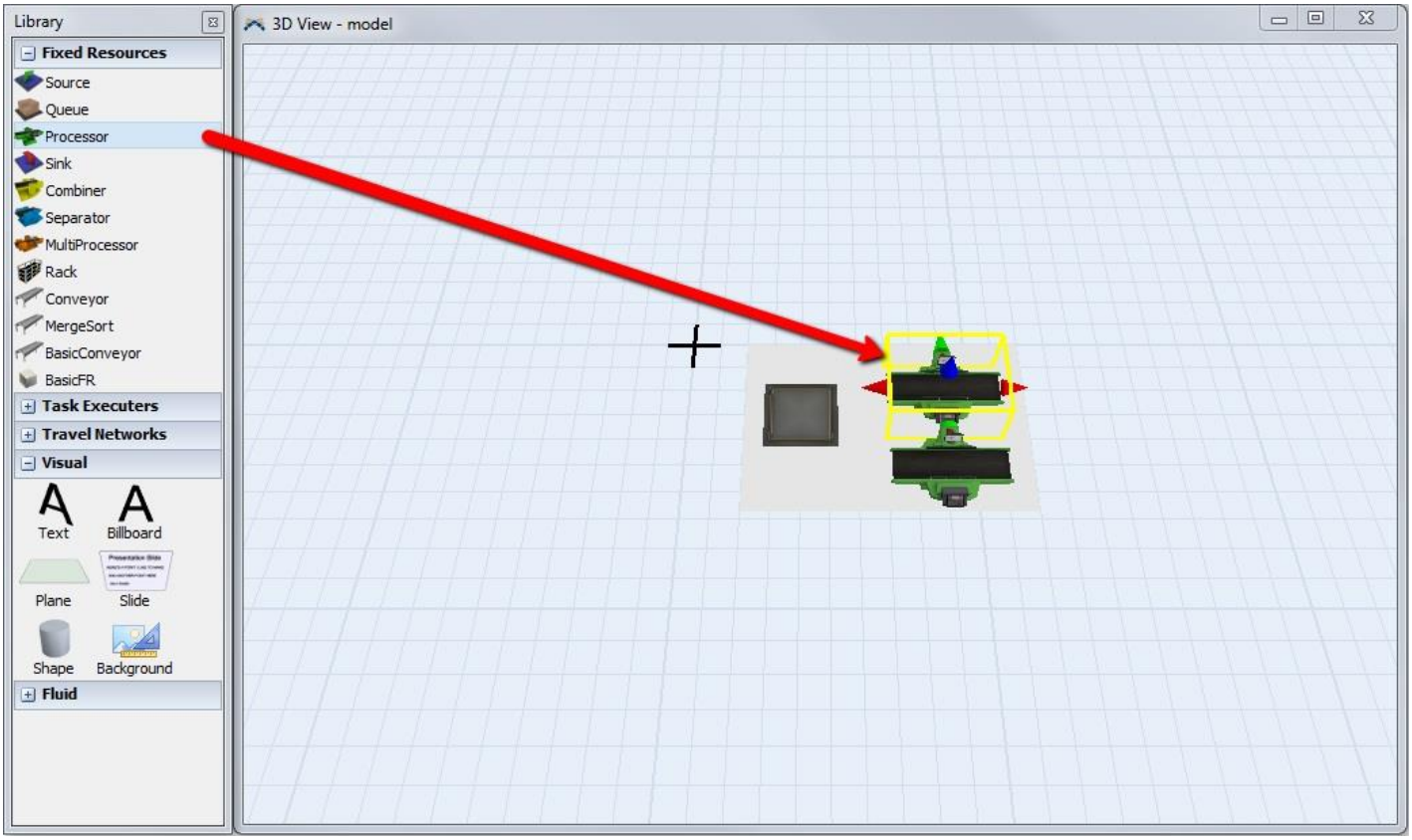
Step 1: Place a VisualTool into the model view

To add a VisualTool to the model, simply drag it from the library onto the model view.



Step 2: Drag-and-drop 1 Queue and 2 Processors into the VisualTool

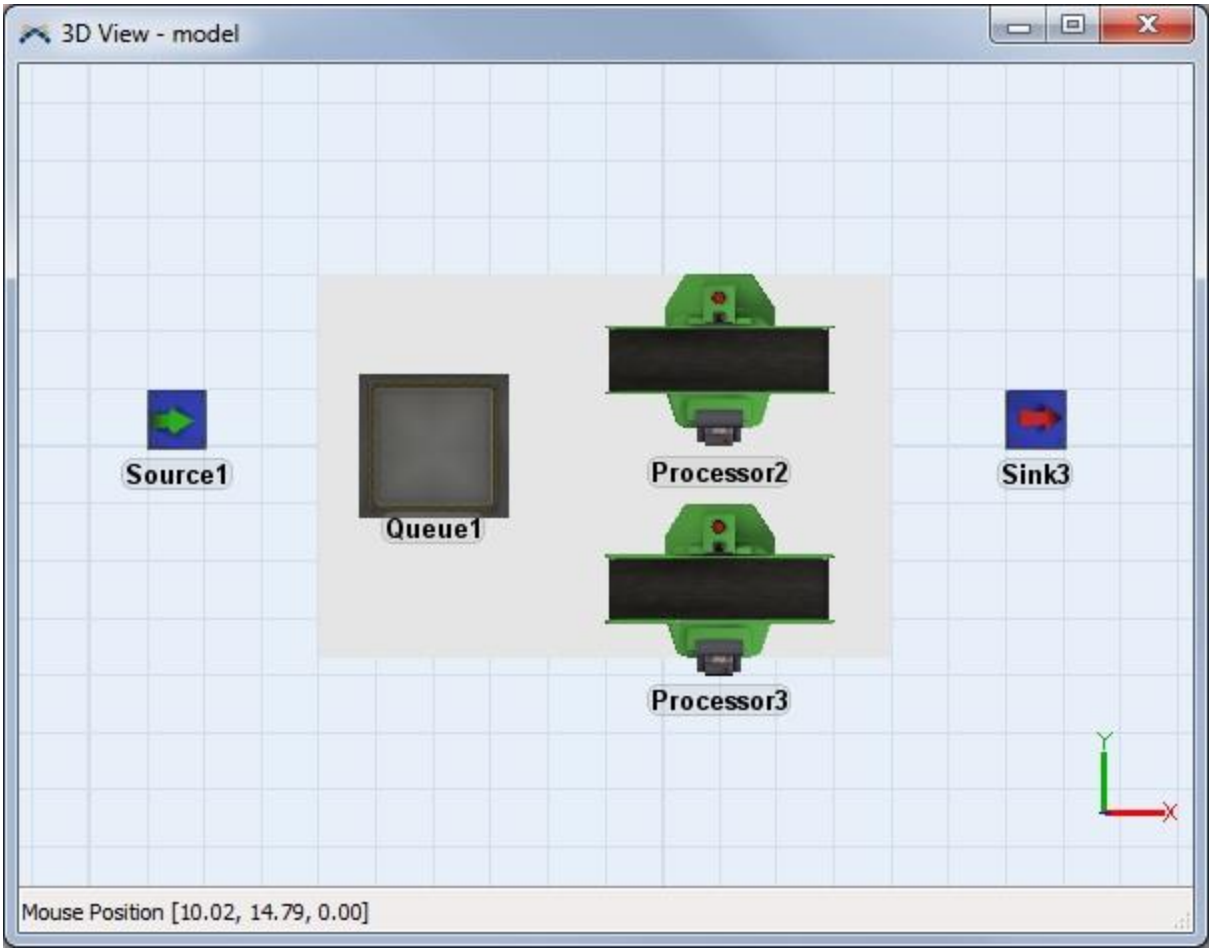
To add objects inside the container simply drag them from the library and place them on the VisualTool object.



When you place an object on the VisualTool it will automatically be place inside the VisualTool object. You can test this by selecting the VisualTool and moving it's location with your mouse. When you move the VisualTool the objects inside will move as well.

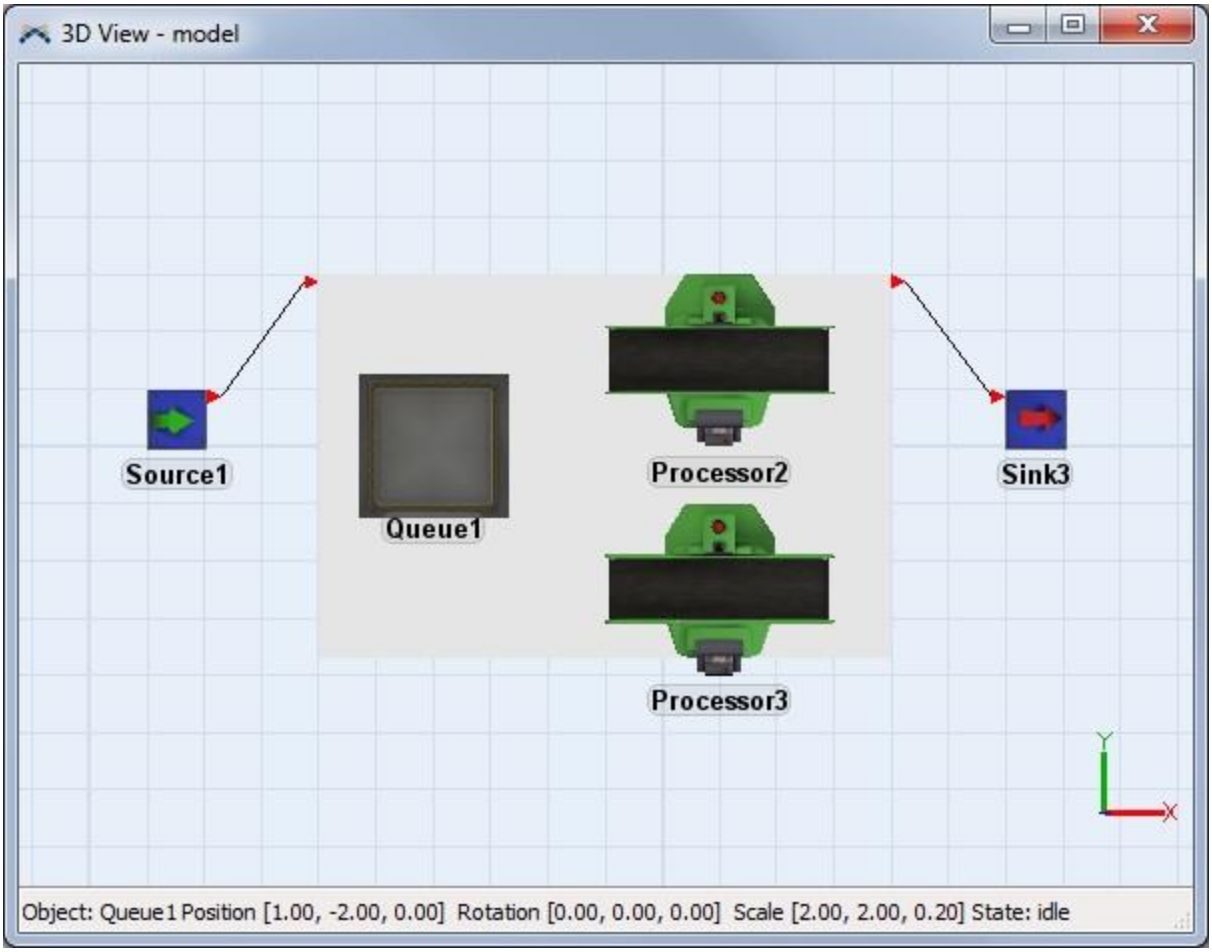
Step 3: Drag out 1 Source and 1 Sink into the model view

When placing the Source and the Sink in the model make sure you do not place them on the VisualTool, you want to make sure they are outside.



Step 4: Connect to Source to the VisualTool, and the VisualTool to the Sink

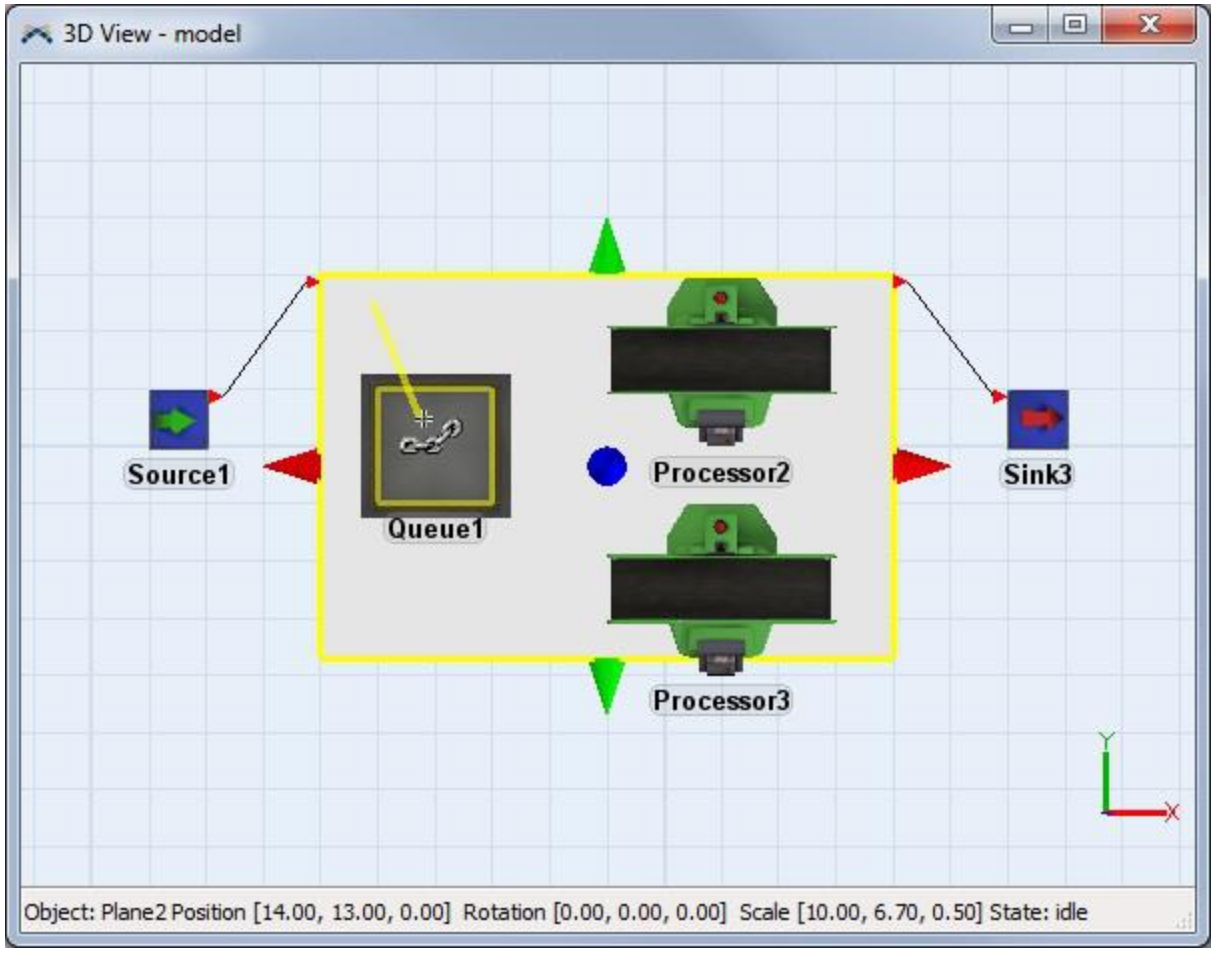
While pressing the "A" key on the keyboard, click-and-drag a connection from the Source to the VisualTool (not the Queue). When you release the left mouse button you will see a connection made between the Source and the VisualTool. Do the same thing to make a connection between the VisualTool and the Sink.



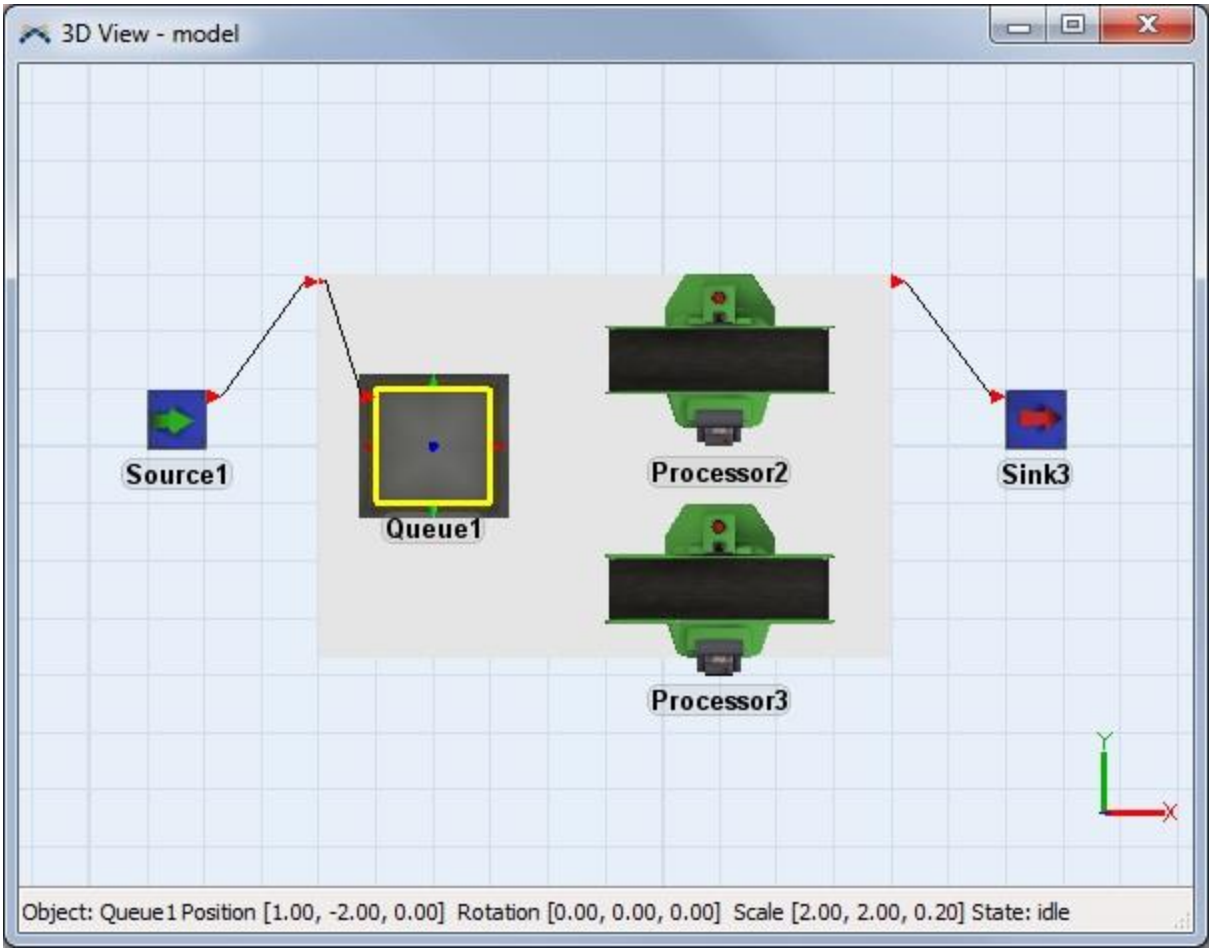
At this point the Source and the Sink are connected to the container (VisualTool). Now we will connect the container to the model inside.

Step 5: Connect the Container to the Queue

Drag a connection from the container to the queue.

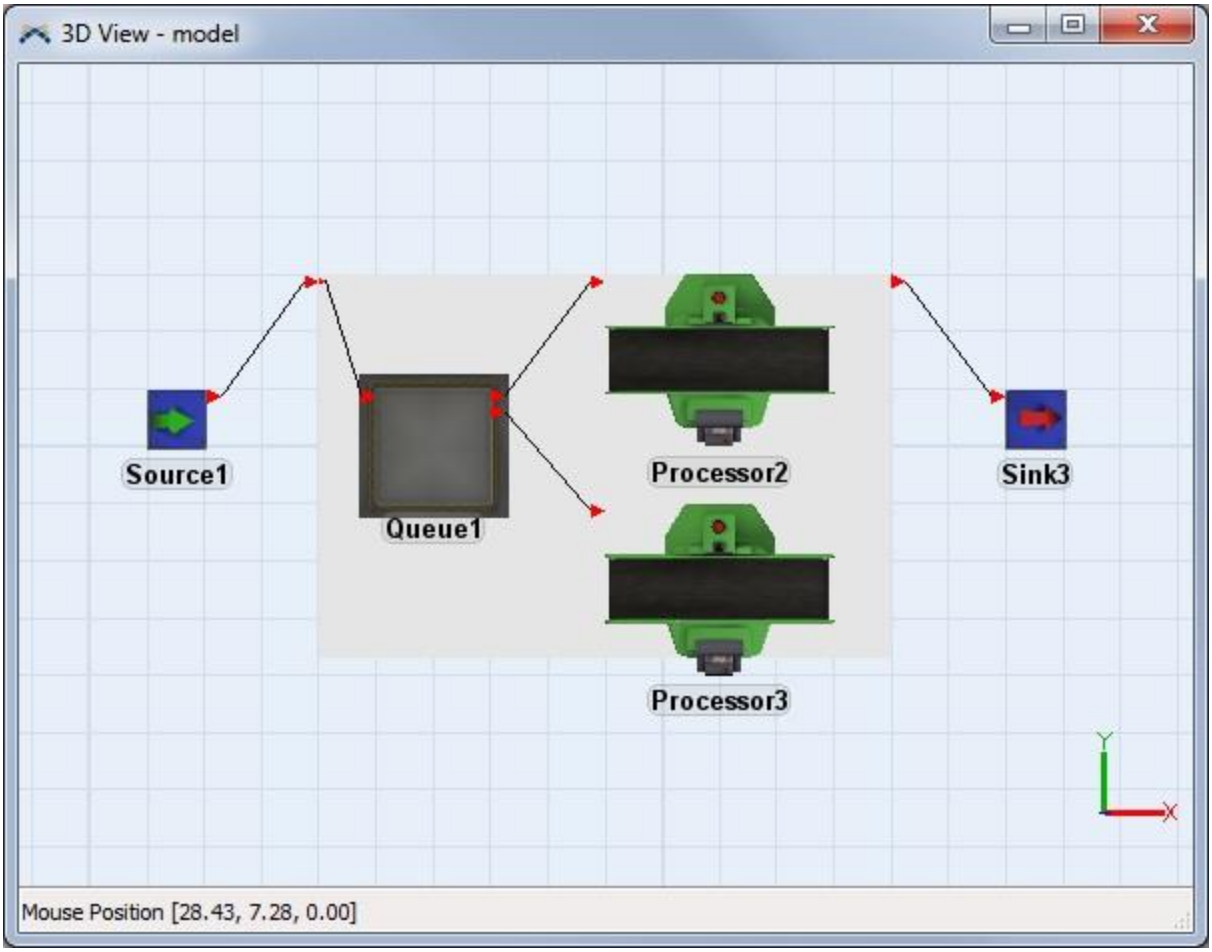


When you release the left mouse button you will see a connection from the internal port (a smaller arrow) to the queue.



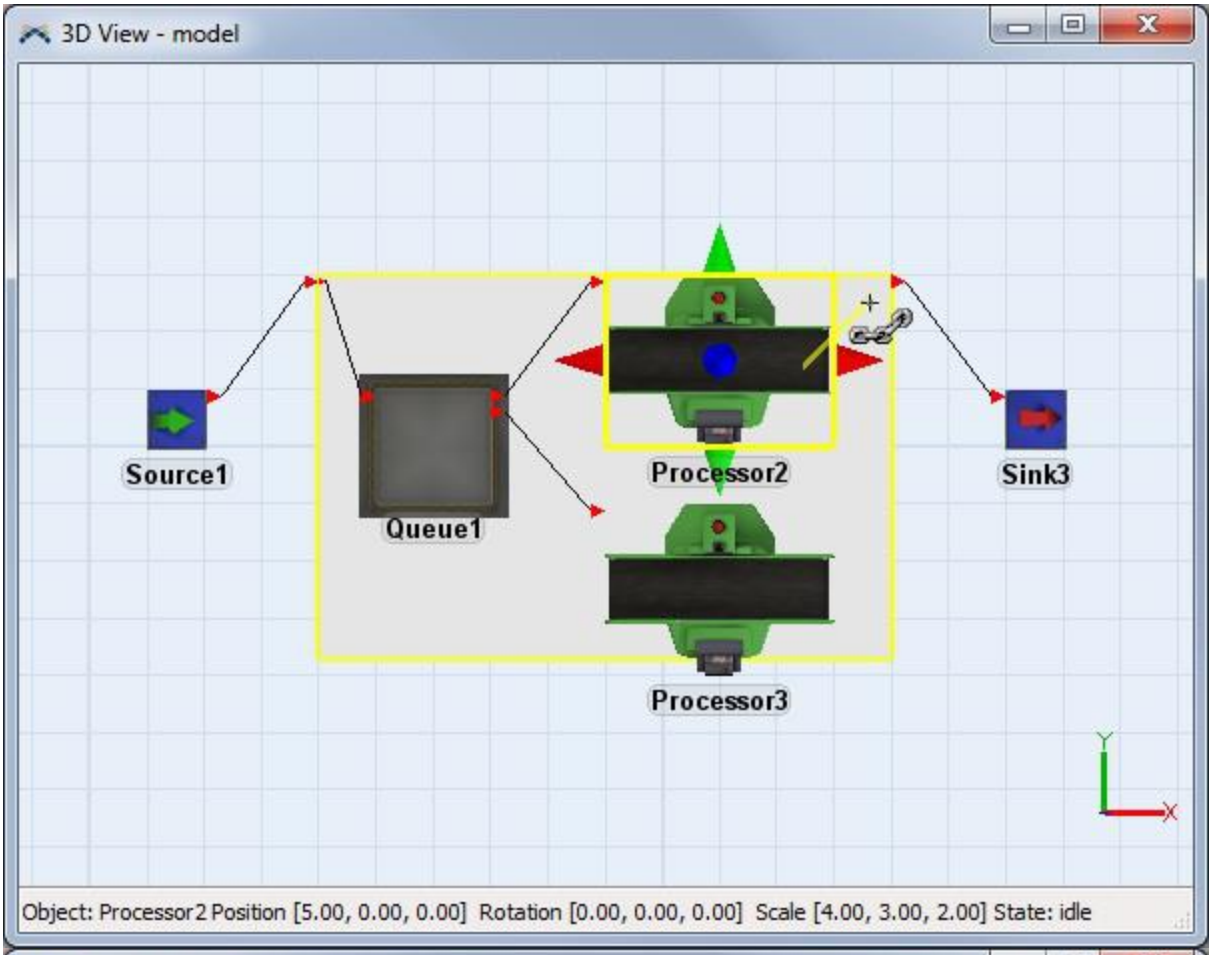
Step 6: Connect the Queue to the Processors

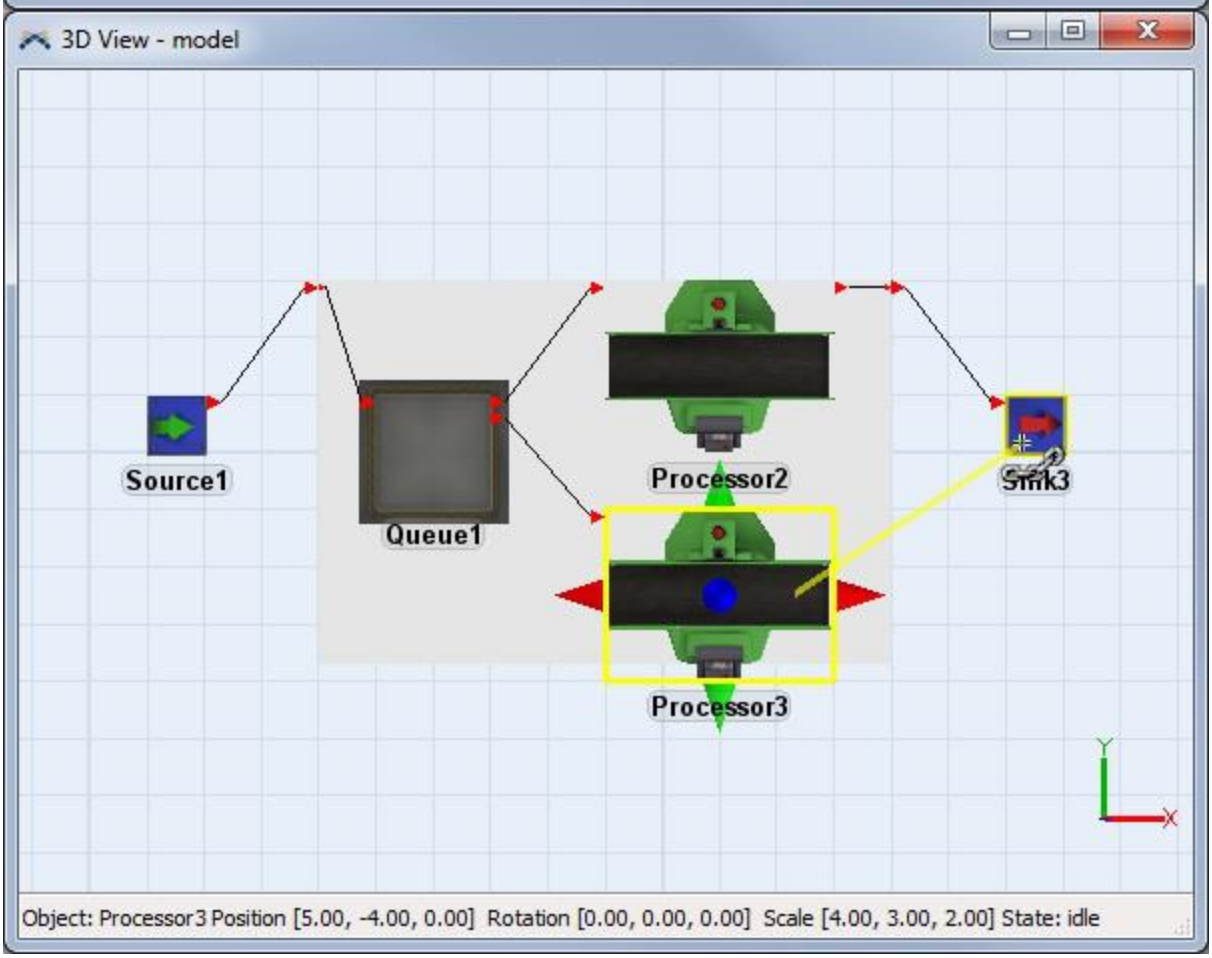
Following the same procedure connect the queue to the 2 processors.

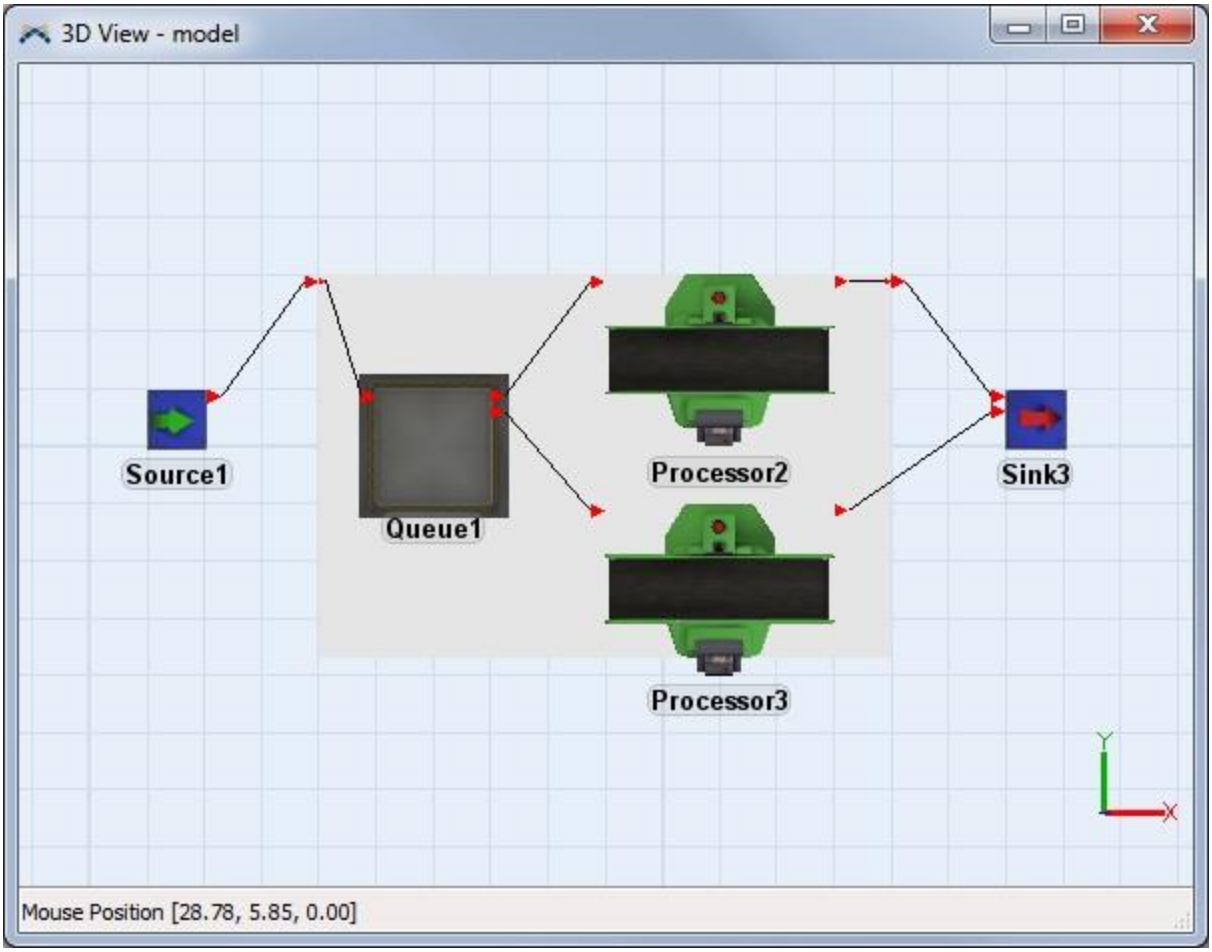


Step 7: Connect the Processors to the container or directly to the Sink

There are 2 ways to connect "into" or "out" of a container. The first way was shown in step 4 when a connection was made from the Source to the container and then from the container to the queue. You can however connect directly from the Processor to the Sink by clicking and dragging a link. For this example the first Processor will be connected to the container which then connects to the Sink, and the second Processor will connect directly to the Sink.


3D View - model
Source1
Queue1
Processor2
Processor3
Sink3
Object: Processor2 Position [5.00, 0.00, 0.00] Rotation [0.00, 0.00, 0.00] Scale [4.00, 3.00, 2.00] State: idle

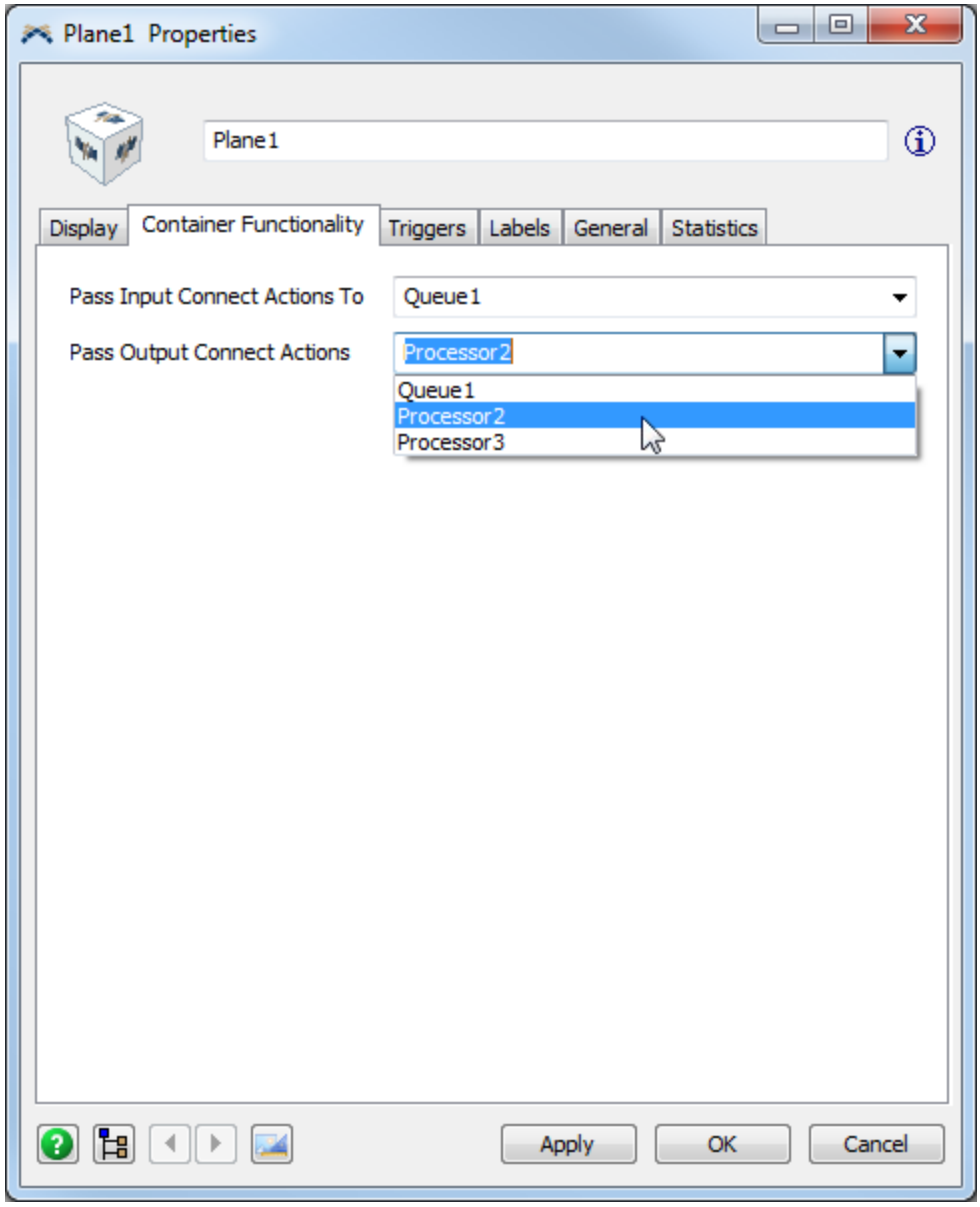

3D View - model
Source1
Queue1
Processor2
Processor3
Sink3
Object: Processor3 Position [5.00, -4.00, 0.00] Rotation [0.00, 0.00, 0.00] Scale [4.00, 3.00, 2.00] State: idle



Step 8: Set the Container Functionality

At this point you now have a functioning container that holds a sub-model. You can leave the container as it is so that the contents are visible and editable, or you can hide the contents so that the container becomes a completely encapsulated black box. To hide the contents of the container, uncheck the **Show Contents** box on the General tab of the VisualTool properties page.

Settings for black box functionality can be found on the Container Functionality tab of the VisualTool properties page. Set the "Pass Input Connect Actions To" to the object that should be connected to the input port of the VisualTool. Set the "Pass Output Connect Actions" to the object that should be connected to the output port of the VisualTool. The internal input and output ports will be connected when an external object is connected to the container's external input or output ports.



You can also use any of the options on the Display tab to present your view of the container as a box, 3D shape, or text. The contents of the container can be viewed at any time by right-clicking on the VisualTool in the ortho window and selecting View > View Contents.

Fluid Library

1. Concepts

Fluid Objects

- **FluidBlender**
- **FluidConveyor**
- **FluidGenerator**
- **FluidMixer**
- **FluidPipe**
- **FluidProcessor**
- **FluidSplitter**
- **FluidTank**
- **FluidTerminator**
- **FluidToItem**
- **ItemToFluid**
- **Ticker**

Fluid Library Concepts

The FlexSim Fluid Library consists of twelve objects. These objects are designed to aid in simulating systems that move material as a fluid. The material does not necessarily need to be liquid, but can be nearly anything that is measured by weight or volume. In contrast, the standard FlexSim library is designed to move material that is in discrete amounts (boxes, pieces, etc). There are fluid objects that are designed to be used as an interface between the other fluid objects and the discrete objects. They allow the modeller to turn flowitems into fluid and fluid into flowitems.

Because of the discrete event nature of FlexSim, the fluid objects are not what might be considered "continuous objects." Instead, they break time down into small sections called "ticks". The length of a tick is called the tick time. At the end of each tick, all of the fluid objects in the model evaluate how much material they received and sent during the tick. Material only moves at the end of each tick. As the tick time gets longer, the model will run faster, but may lose accuracy. As the tick time gets lower, the model will slow down, but may be more accurate. It is the modeller's responsibility to make sure that the tick time for the model is small enough that the simulation is still accurate.

The amount of material that moves between objects at the end of each tick is based on the input and output rates defined on each object, as well as the amount of material that is available to move and the amount of space available to move it to. The fluid objects have a standard way of moving material between themselves. The input and output rates are each defined using three values: the object's maximum total rate, the maximum port rate, and a scaling factor for each port. The object's maximum rate is the maximum amount that will be allowed in or out through all of the ports combined in a single time unit (not a single tick). Because fluid movement is evaluated one port at a time, this value may be used to stop certain ports from sending or receiving. The maximum port rate is the maximum amount of material that will be allowed in or out any one port in a single time unit. This value applies to all of the input or output ports on the object. Each individual port is then assigned a scale factor. This scale factor is multiplied by the port rate to calculate the maximum rate for that specific port. These scale factors are used to restrict flow through one or more ports without affecting the maximum flow through the others. These three values are applied separately to the input and output ports.

All of the fluid objects use this system to transfer material. However, because of the specialized nature of some of the objects, not all of these values may be available to the modeller for editing. Instead, some limited subset of the values will be presented and the object will maintain the other variables behind the scene. All of the values that the user can edit are presented in the GUI's for each object, and will be discussed in more detail later.

Each object also keeps track of a value called the Product ID. This is a number that is used to identify the type of material that the object is currently holding. The product may contain sub-components. This is a list of all of the possible sub-components in the model. The list is recorded as a set of percentages of each sub-component. The Product ID number of an object does not represent any of the sub-components. It is simply a number that the modeller assigns to a specific material. The Product ID of the objects, as well as the sub-components percentages will always be maintained by the fluid objects. The user only has to specify initial values on the objects that create fluid.

Most of the fluid objects have a system for displaying the current content of the objects as a percentage of the maximum content. This system is a colored bar called the level display or level indicator. The bar consists of two layers. One is dark gray and is generally drawn above the other. This represents the amount of empty space in the object. The other is the color of the object and represents the current content. If a bar is displayed as being fully gray, that object is empty. If a bar is fully colored, that object is full.

The level indicator bar can be moved, resized and rotated for each object in the model. The modeller can also state whether the bar is rectangular or cylindrical. The size of the bar is measured as a percentage (from 0-1) of the size of the object. The location is relative to the size of the object as well. The point (0,0,0) is one corner of the object's bounding box while (1,1,1) is the opposite corner. This flexibility allows the modeller to position the level indicator bar in such a way that it appears to be part of the object's 3D shape.

Modeling Tools

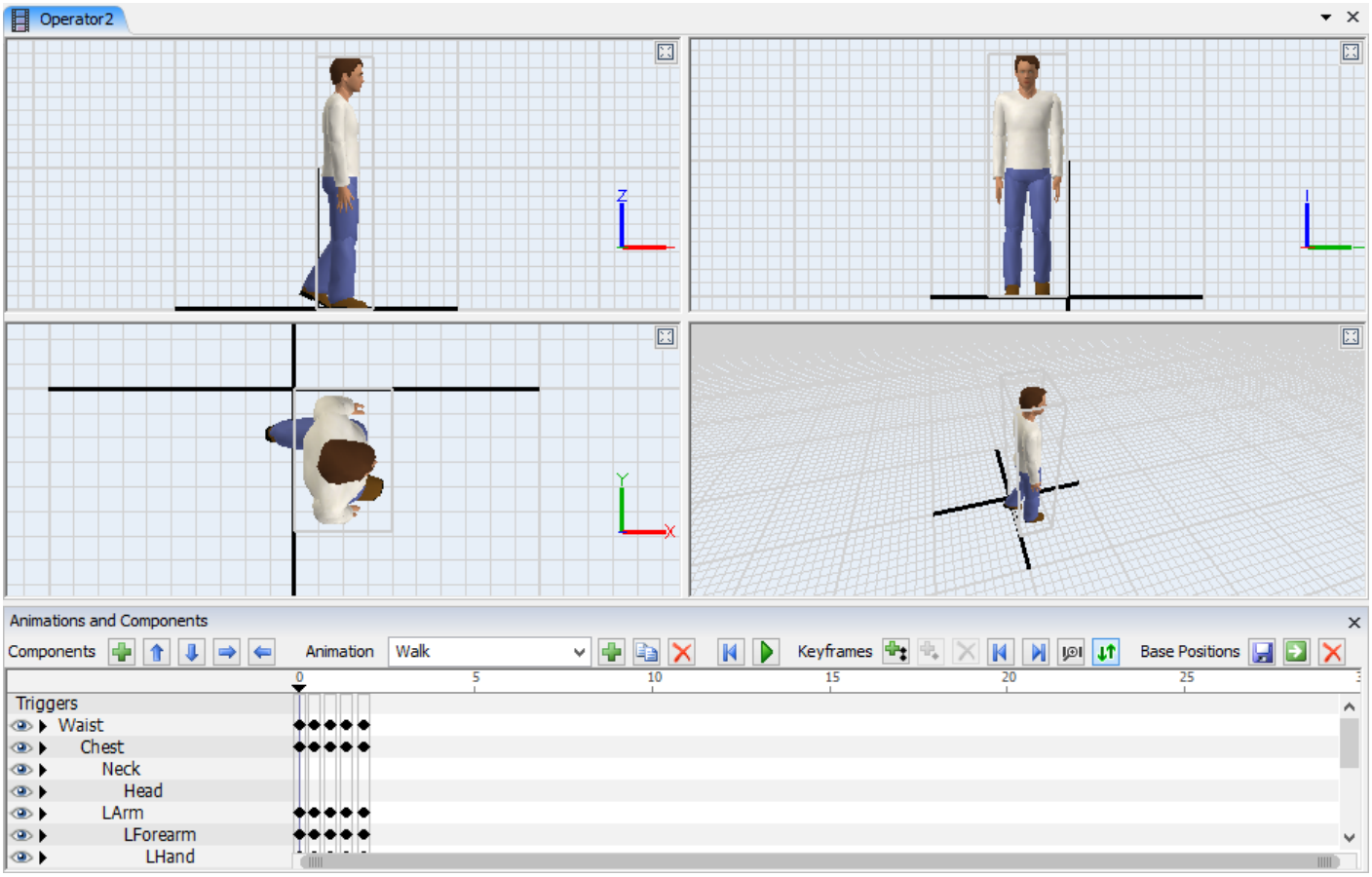
This section contains information about different modeling tools that you will use in building, configuring, and obtaining results from your model. Many of these tools can be accessed through the Tools menu, while others are accessed from other areas.

1. **Animation Creator**
2. **AVI Maker**
3. **Event List**
4. **Event Log**
5. **Excel Interface**
6. **Flowitem Bin**
7. **Global Tables**
8. **Global Task Sequences**
9. **Global Variables**
10. **Graphical User Interfaces**
11. **Import Media Files**
12. **Model Background**
13. **Model Triggers**
14. **MTBF/MTTR**
15. **Presentation Builder**
16. **Script Console**
17. **Time Tables**
18. **Tracked Variables**
19. **User Commands**
20. **User Events**
21. **Visio Import**

Animation Creator

1. Concepts
2. Example
3. Reference

Animation Creator Concepts



Topics

- **Using Animations**
- **Components**
- **Library Icon Grid**
- **Repeating Animations**
- **Time Basis**
- **Animation Variables**
- **Keyframe Triggers**
- **Surrogates**
- **Commands**

The Animation Creator can be accessed through either the right click menu in the 3D View **Edit** > **Animations** or through the General page of the object's Properties Window.

The Animation Creator allows you to create and save custom animations for FlexSim objects and Flowitems in your model.

In FlexSim, animations are made up of components and keyframes. Components are just 3D shapes inside your object. Keyframes are values that are saved for different properties of your components. The Animation will automatically tween between keyframes to give the animation smooth movement.

An object many have any number of components and each component may have any number of keyframes. Keyframes can be used to specify the position, size, rotation, color and shape frame of the component. Animations can also have their own custom triggers that fire at certain keyframes in the animation. These options allow you to add really sophisticated animations to your objects.

Using Animations

Animations can be used at any point in your model. An example of an animation is the Operator walking. The Operator is not one 3D shape, rather it is made up of 16 different shapes. These shapes, or components, each have their own set of keyframes that change their position and rotation as the Operator walks. The walking animation is based upon distance traveled, rather than on time.

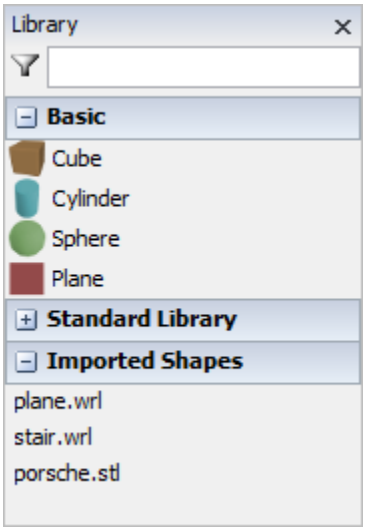
Animations can be used to make a processor look more realistic, or to show a flowitem changing in a process. You can start and stop these custom animations through Flexscript commands.

Components

Components are 3D shapes inside your object. They can be manipulated using keyframes to change their position, size, rotation etc. In the Operator, as seen above, there are components of Waist, Chest, Neck, Head etc.

Library Icon Grid

When the Animation Creator is active, the Library Icon Grid changes to display a list of shapes:



Shapes that have been imported into your model will appear at the bottom of the Library. You can drag and drop shapes from the Library onto one of the 3D views to add a component to your object with that shape.

You can filter the list of shapes by entering text into the field.

Repeating Animations

Animations can be repeated using the following types:

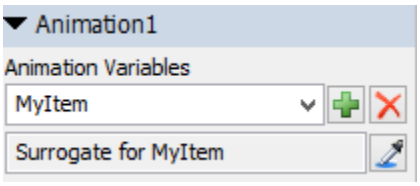
- **Do Not Repeat -** The animation will stop as soon as it is complete.
- **Repeat Indefinitely -** Causes the animation to loop back to the beginning forever.
- **Repeat Set Number -** Causes the animation to loop back to the beginning X number of times.
- **Repeat After Time -** This will cause the animation to loop back to the beginning after X amount of time (or distance) regardless of whether or not the animation has completely. For example, an animation taking 10 seconds with a repeat time of 5 seconds will only play half of its animation.
- **Time After Animation End -** Once the animation has ended, the animation will wait X amount of time (or distance) before starting again from the beginning.

Time Basis

Time Based - When an animation is set as Time Based, the animation will play based on the simulation run speed. The numbers displayed on the timeline are in model time units (as defined in your Model Settings window).

Travel-Distance Based - When an animation is set as Travel-Distance Based, the animation will play as the object moves in 3D space. The numbers displayed on the timeline are model length units (as defined in your Model Settings window), rather than time. An example of a Travel-Distance Based animation is the Operator's Walk animation.

Animation Variables



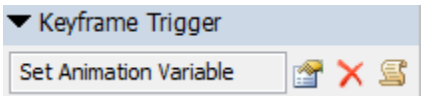
Animation variables can be added to any animation. These variables can be used to dynamically change values on your animation. For example, changing the position or rotation of a component, changing the component's color, or even changing a shape frame. Animation Variables are pointers to components or parameters of components. In the above image, MyItem is an animation variable pointing to the Surrogate for component MyItem. The variable can be set through this interface, or through code by using the setanimationvar() command. Animation Variables can point to:

- Components
- Surrogates
- Keyframe Times
- Time Gaps
- Component Keyframes (Position, Size, Rotation, Color, Shape Frame)

Animation variables pointing to a component are static variables. They are merely available for getting quick reference to a component through getanimationvar. Calling setanimationvar to a variable pointing to a component will have no affect.

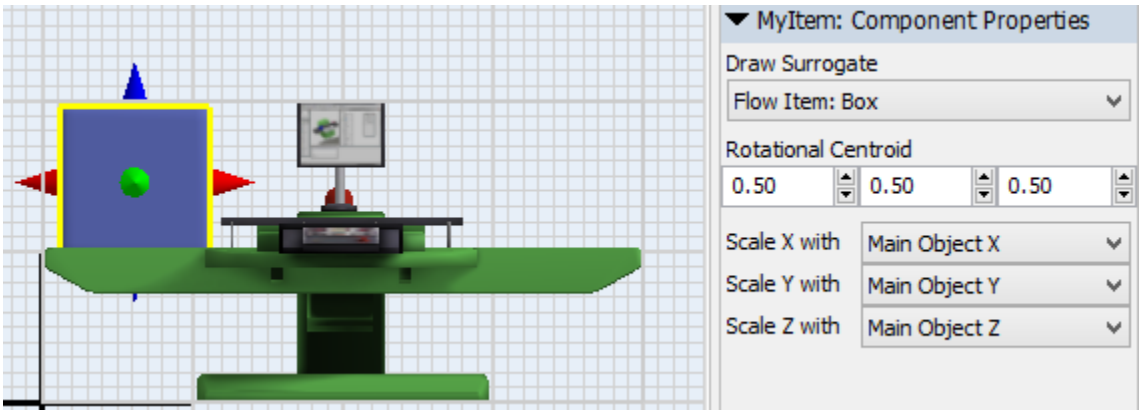
If you want to dynamically change the length of your animation based on parameters in your model at run time, you can create an animation variable that points to a time gap, and then set that animation variable through a keyframe trigger or from a trigger in your model.

Keyframe Triggers



Triggers can be added to animations that will fire when the animation gets to a specific (or distance). They are added through the timeline. Triggers allow you to dynamically change or update animation variables, stop the animation, change some parameter in your model, or execute any other Flexscript code.

Surrogates



Surrogates are used when you want to replace a component in your animation dynamically, with an object in the model. For instance, as seen in the image above we have a component called MyItem that currently

has a Surrogate drawn as a box. The shape that is being drawn for the surrogate is not important and will not show up in the model. It is purefly for the modeler's convenience.

When you add a Surrogate, it will ask you to create an animation variable. This animation variable can be set to an object in the model that will be drawn in place of this component. Once you have made a surrogate, you can specify it's Rotational Centroid and determine how it will scale the object in the model.

Surrogates can be set by using the setanimationvar() command from somewhere in your model, or you can specify a keyframe trigger on your animation that will set the animation variable.

For an example of using a surrogate, see the Surrogate Example.

Commands

The following commands deal with animations:

startanimation(obj, animation) - Starts an animation on obj. Animation can be either a string value that is the name of the animation, or it can be a number which is the animation rank.

stopanimation(obj, animation) - Stops an animation on obj. Animation can be either a string value that is the name of the animation, or it can be a number which is the animation rank.

resumeanimation(obj, animation) - Resumes an animation on obj that was previously stopped. Animation can be either a string value that is the name of the animation, or it can be a number which is the animation rank.

getanimationvar(obj, animation, "varname") - Returns the value of an animation variable.

setanimationvar(obj, animation, "varname", value) - Sets the value of an animation variable. Value can be a number or object and is based upon what the animation variable is linked to.

Animation Creator Example

Examples

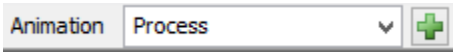
- **Creating an Animation**
- **Keyframe Triggers**
- **Surrogates**

Creating an Animation

As an example, we will create a simple animation for an operator that he will use while processing an item at a Processor.

Step 1

- Create an Operator.
- Right click on the Processor and select **Edit > Animations...**. This will open the Animation Creator.
- Create a new animation by clicking the button next to the Animation combobox. Rename the animation to "Process"./li>



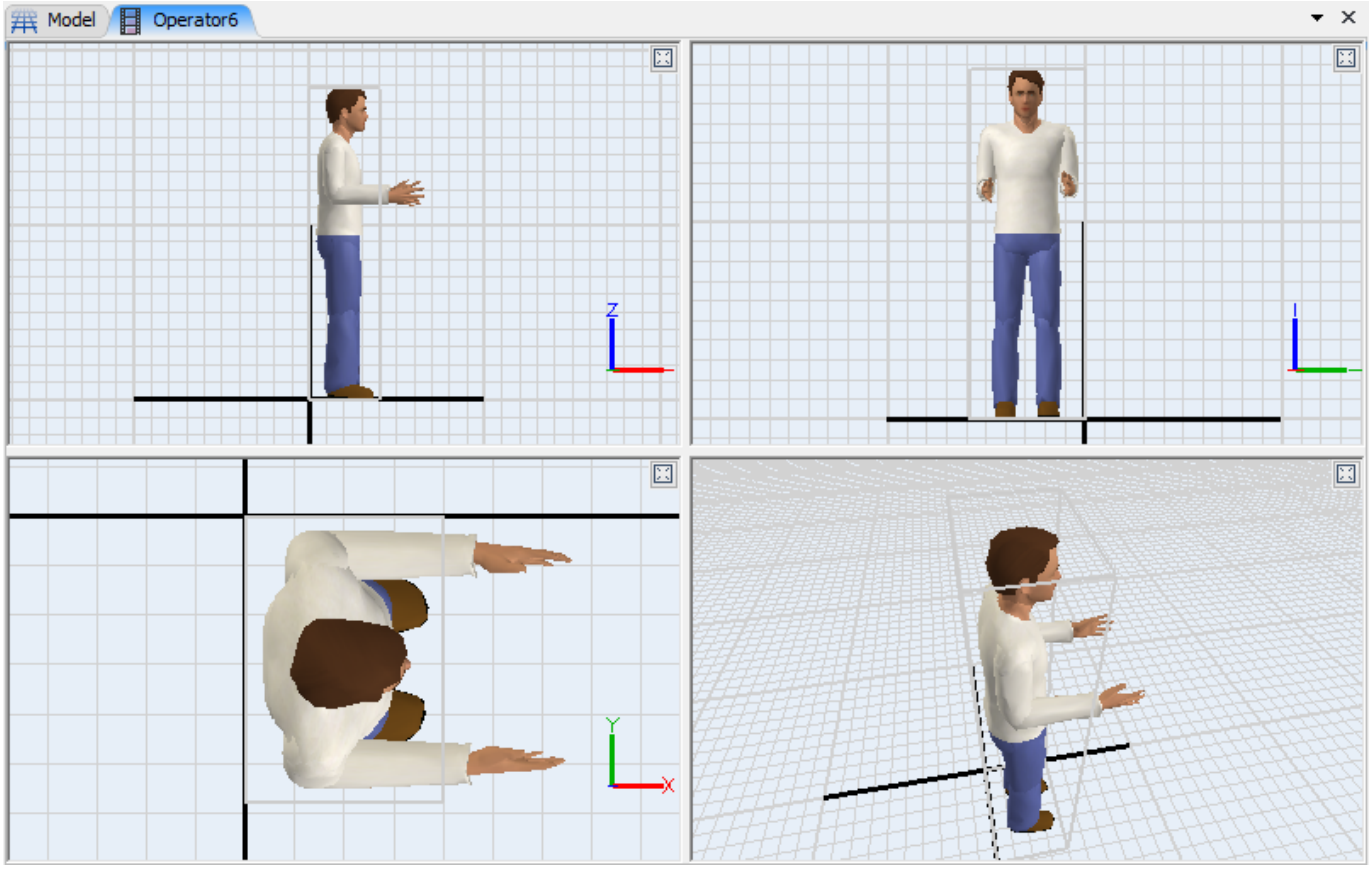
You now have a "blank slate" on which you can create a new animation, customized to your needs. To keep things simple, you are going to create an animation with only 3 keyframes and some very simple arm movement so that you can learn the basics of the Animation Creator.

You'll notice in the Animations and Components window that there is already multiple components that make up the Operator (Waist, Chest, etc). Components that are inside of other components (indicated by the indentation) are subject to changes made by their parent component. For example, click LArm, then click and drag on it in any of the 3D views to move it. Or change the X, Y, or Z values in the Quick Properties. As you move the LArm, you will see that the LForearm and the LHand move along with the LArm. The same is true for rotating and scaling, though there are some different options for scaling that will be discuss later.

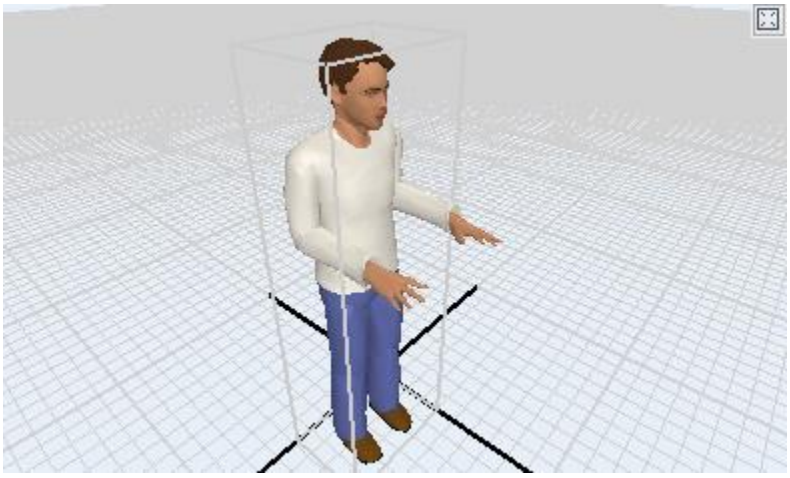
When making changes to objects in the Animation Creator, it is usually wise to start from the outside (parent components), and work your way through all of the child components. If the Operator's whole arm is going to change positions, start with the arm, then the forearm, then the hand. Before we move on, if you have messed up how your Operator looks, click the Go To Base Positions to revert to the normal standing operator.

Step 2

- Click the button to make sure that you are working at the beginning of the timeline.
- Click LForearm in the components list, and change RX ( ) in the Quick Properties to -90. Do the same to the RForearm.



- Click LHand in the tree, and change RZ to 0. Change the RHand RZ to -90. He now has both arms out, with his palms facing down. This will be our starting position.



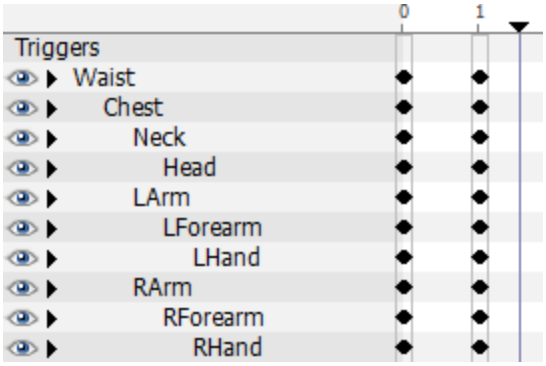
- Now that our positions are set, click button to add a keyframe to time 0.

Step 3

You'll notice that the cursor on the timeline jump ahead one second when we hit the button. This allows you to quickly make adjustments and add keyframes all at once without overwriting any changes. You can then change the timing of the keyframes by clicking and dragging them along the timeline.

We want to make sure our animation is seamless when it repeats. The easiest way to do this is to create another keyframe, a duplicate of the first.

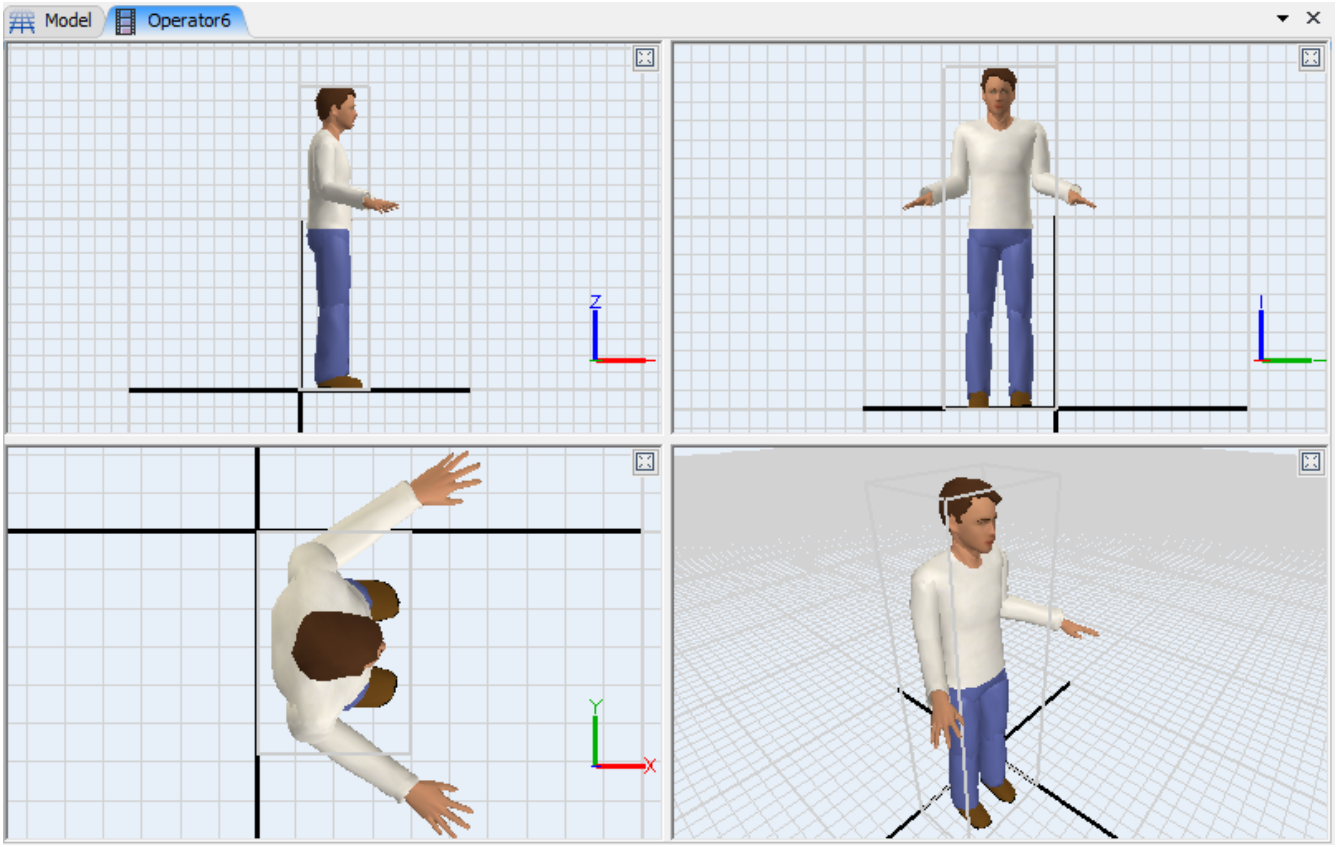
- Click the button to add a keyframe to time 1.



Step 4

We will now insert a keyframe to the middle of our animation.

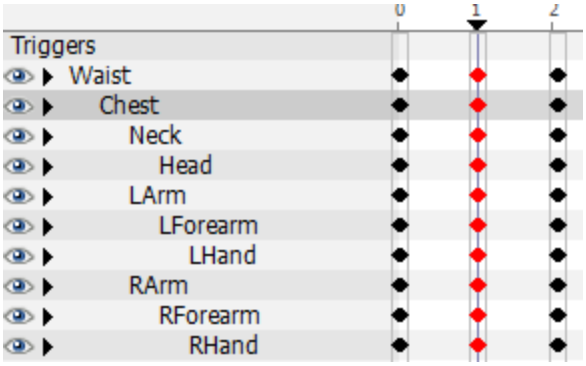
- Move the cursor to somewhere inbetween 0 and 1 by clicking on the timeline.
- Click LForearm, and change the RZ to 35, and the RX to -75. Do the same with RForearm, but change RZ to -35.



- Click the button to add a keyframe to the timeline.

Now that we have our all 3 keyframes, we want to slow the animation down slightly.

- Click between two of the vertical keyframes at time 1 (clicking on a keyframe will only select that one keyframe). The column of keyframes will turn red.
- Drag the keyframes from time 1 to time 2.
- Click and drag your second keyframe (time 0.5) and move that to time 1.



You can now preview the animation by clicking the button.

Step 5

We want to make sure the animation repeats as long as the Operator is processing an item. You can change the Repeat type for the animation in the Quick Properties under the Process panel.

- Set the Repeat Type to **Repeat Indefinitely**.

Step 6

Now that you've created an animation, you can have your Operator run that animation when he processes an item at a Processor. All you need to do is call the following Flexscript code:

```
startanimation(operator, "Process");
```

To stop the animation call:

```
stopanimation(operator, "Process");
```

Keyframe Triggers

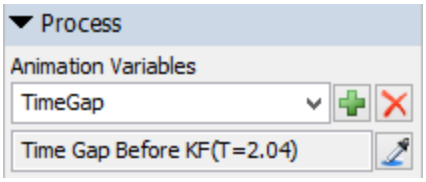
In this example we'll show you how to create a dynamic animation using Keyframe Triggers. If you have no already done so, complete the above example on Creating An Animation before starting this example.

In the Operator's Animations, make sure you are editing the "Process" animation.

Add an Animation Variable

We'll add an animation variable to our animation that points to the Time Gap between the 2nd and 3rd keyframes. This will allow us to optionally slow down the animation based on a given parameter in the model.

- Under the Process panel in the Quick Properties, add an Animation Variable by pressing the .
- Set the name of the new variable to TimeGap.
- Click on the to enter "Sample" mode. Mouse over the gap between keyframes 2 and 3 in the timeline. The cursor should read something like "Time Gab Before KF(T=2.0)". Click the mouse to sample this value.

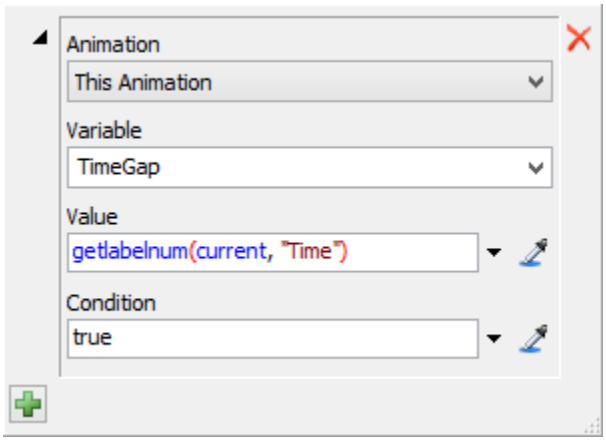


Add a Keyframe Trigger

- Click on Triggers line in the timeline above the 2nd keyframe.
- Click on the to add a keyframe trigger.

A new panel will appear in the Quick Properties called **Keyframe Trigger**.

- Click the and select **Set Animation Variable**.
- Enter the following values in the popup:



Add Time Label

The last step is to add a label to your Operator called "Time". You can then change this value during your model run and the animation will dynamically change the distance between keyframe 2 and 3, effectively slowing down or speeding up that section of the animation.

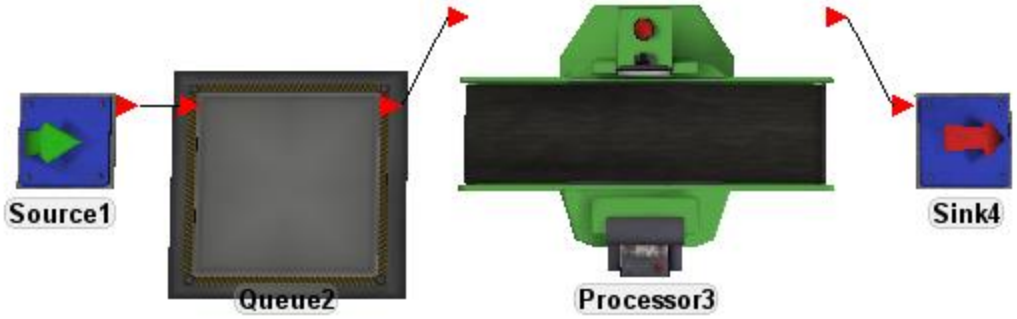
This same method can be used for changing position, size, rotation, color and shape frames of components.

Surrogates

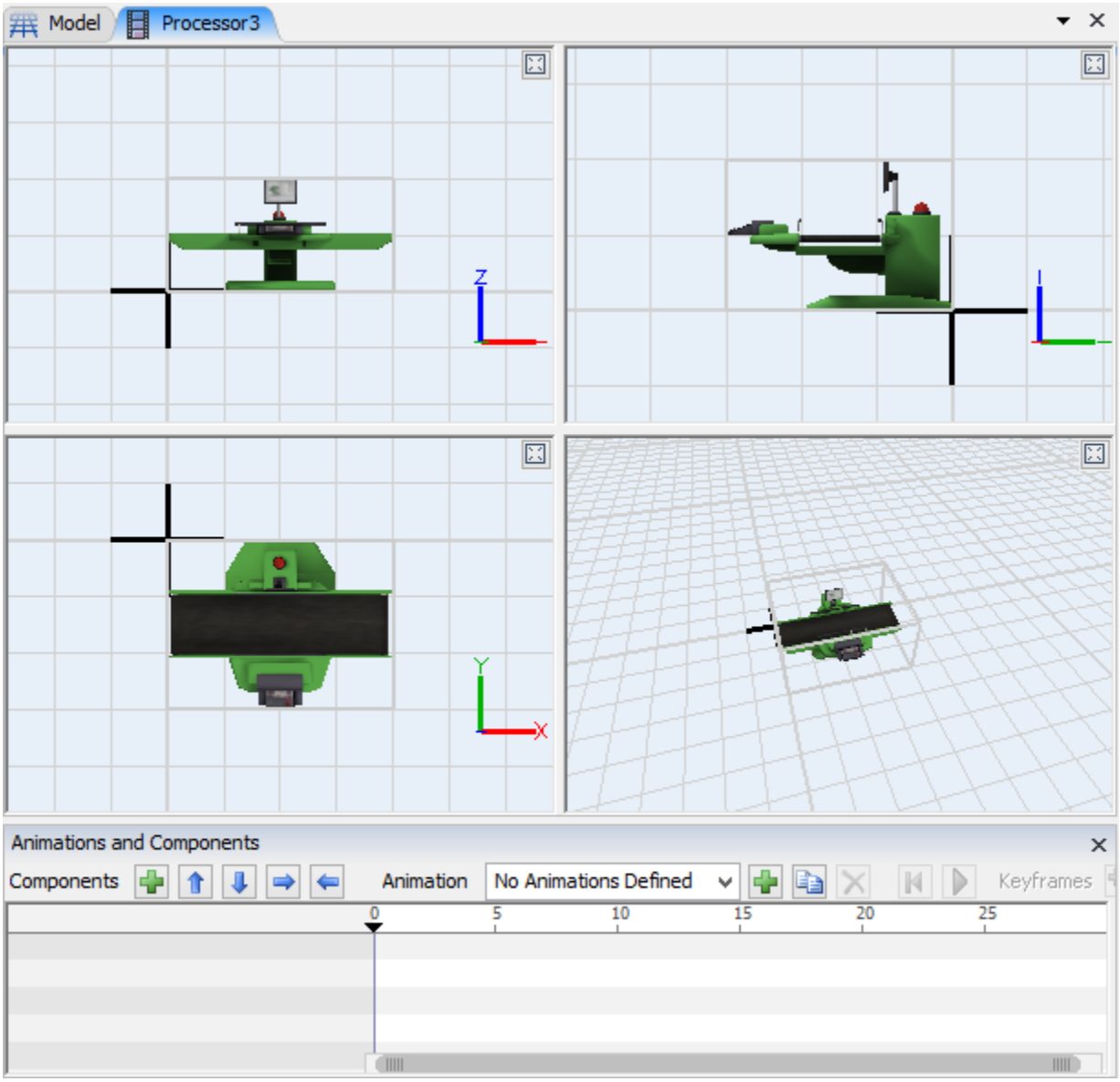
Let's say we want to customize what our flowitem does when it enters a Processor rather than just moving across it. The easiest way to do this is to create a custom animation on the Processor using a Surrogate.

Create the Model

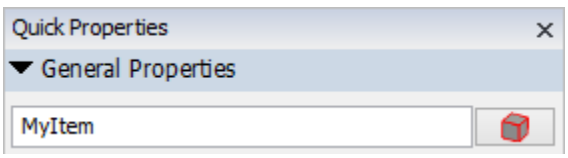
Create the following simple mode:



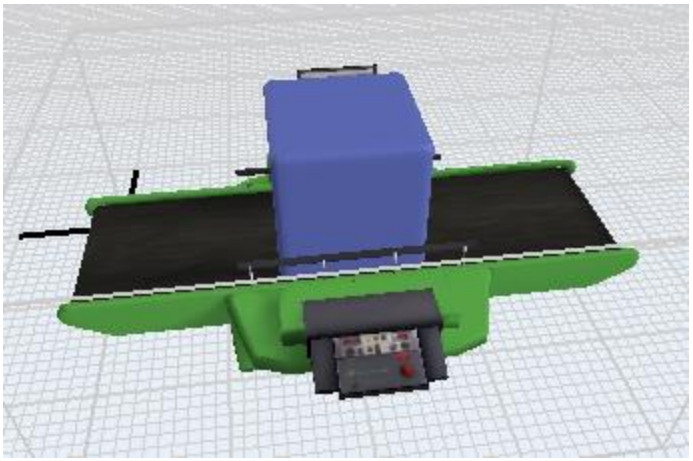
Right click on the Processor and select **Edit > Animations...**. This will open the Animation Creator.



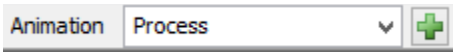
In the Animations and Components window, press the + to add a new component.
In the Quick Properties window, set the object name to *MyItem*.



In the 3D views, position the item so it is on the center of the Processor.

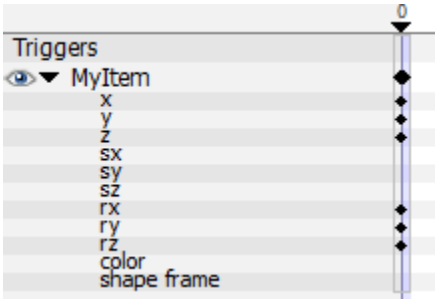


We now need to add an animation to the Processor. In the Animations and Components window, press the next to the Animation combobox. Rename the animation to "Process".



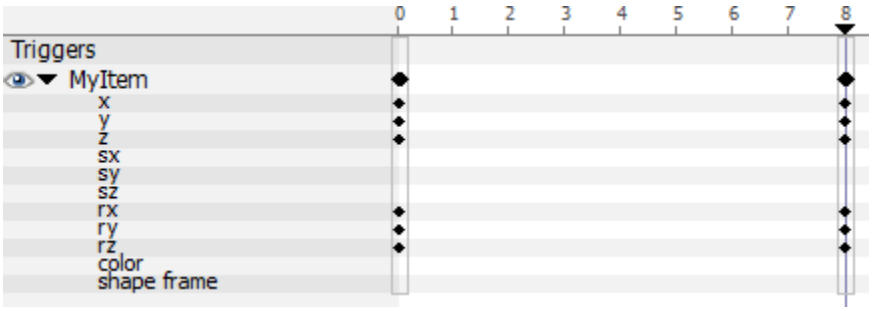
Creating Keyframes

We will create two keyframes in the MyItem component. Since the component is where we want it when the animation starts, we can go ahead and create a keyframe now. Press the button to add a keyframe to MyItem. If you open the component's list of properties, you'll see this adds keyframes to the position and rotation of the object. In our case, we're only going to modify the z rotation, however, it won't cause issues having these extra keyframes.



Move the time cursor in the timeline over to the 8 by either click on the 8 or anywhere beneath it. Now we can reposition MyItem. In our case we want to rotate the component 360 degrees about the z axis. You can rotate it in the 3D view by right clicking on the blue arrow and dragging up or down, or you can manually type in 360 into Z Rotation field of the Quick Properties.

Now click the button to add a new keyframe to MyItem.



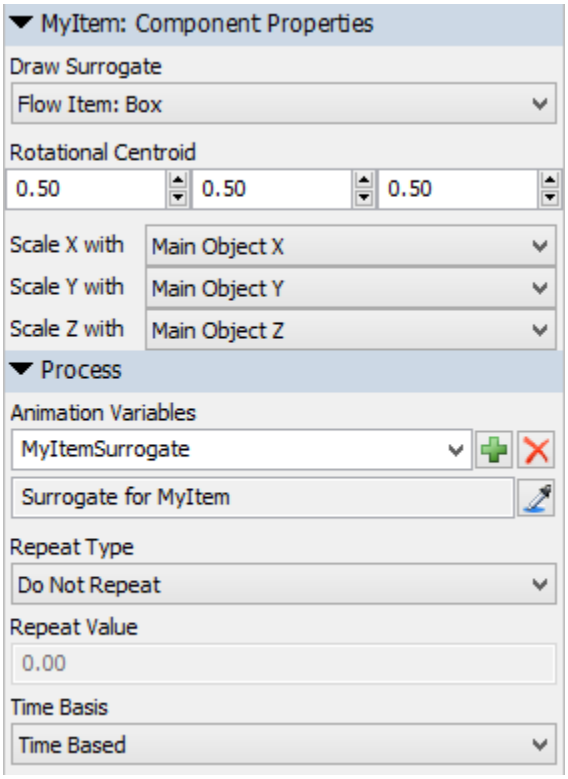
Press the to play your animation. You should see MyItem continuously spin about it's z axis.

Creating a Surrogate

Having a cube spin around on our Processor is great, but it doesn't look the same as our flowitem that will be processed at this Processor. To accomplish that, we will have MyItem draw a surrogate.

If MyItem is not selected, click on it in the 3D view or in the Animations and Components window. In the Quick Properties, you'll see a panel labeled *MyItem: Component Properties*. Set the **Draw Surrogate** to *Flowitem: Box*.

Click the Add button to tie the surrogate to an Animation Variable called "MyItemSurrogate".



Run the Animation

Go back to your model. You'll notice that the box does not appear. When we assigned it to draw a surrogate, it sets the component to be hidden until we assign it a valid 3D object.

Open the Processor's Properties page, and under the Trigger's tab, press the next to the **OnEntry trigger** to open the code editor. Enter the following code:

```
setanimationvar(current, "Process", "MyItemSurrogate", item);

startanimation(current, "Process");
```

Edit the OnExit trigger with the following code:

```
stopanimation(current, "Process");
```

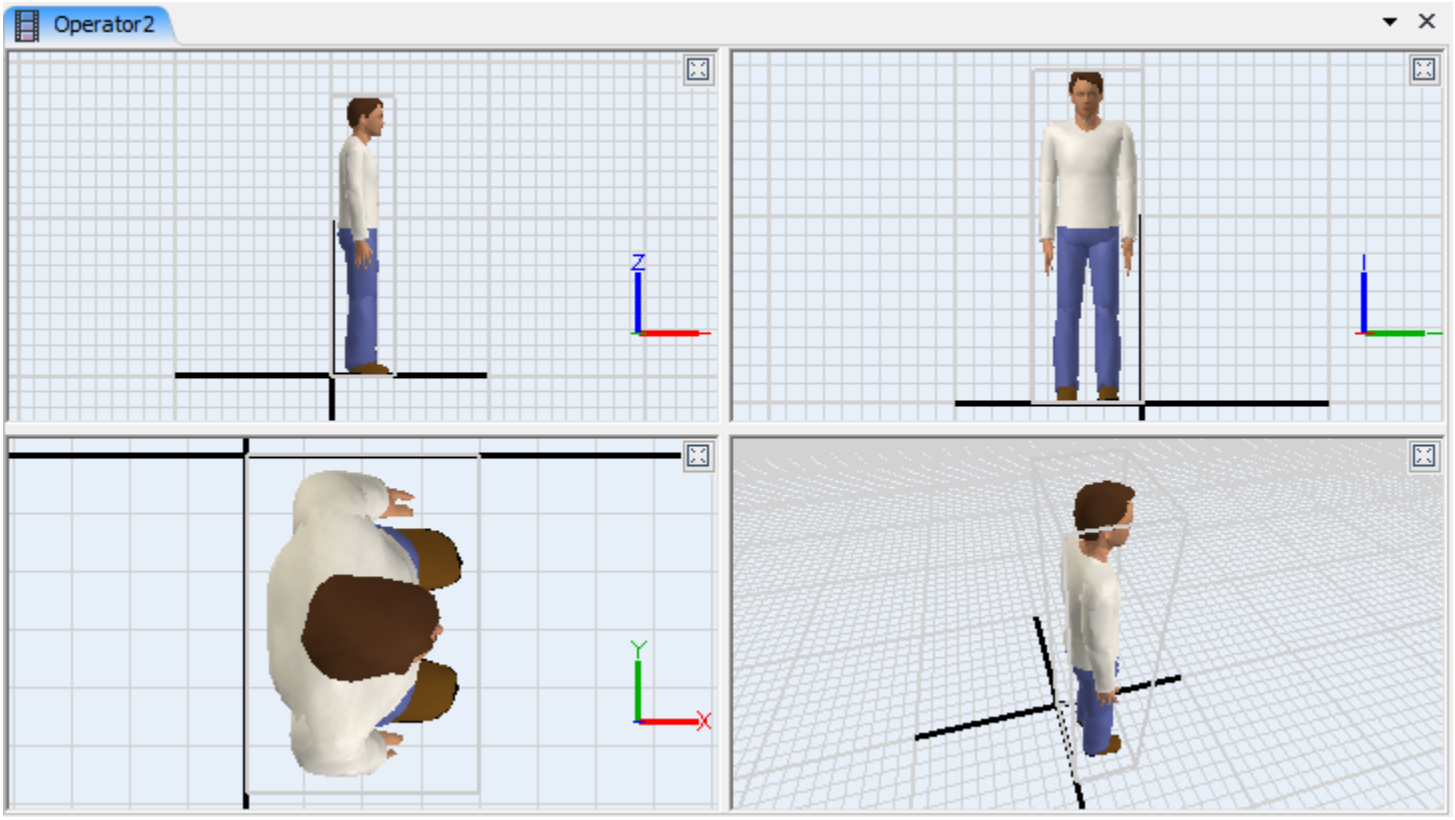
Lastly, we want to hide the item that's in the processor as our surrogate will be acting as the item. Under the Processor's **General** tab uncheck the **Show Contents** button.

Now Reset and Run the model to see what your animation looks like.

Currently we are using the Source's default Flowitem class, the Box. However, you can change the Source to flowitems with any 3D shape and that shape will display on the Processor, spinning.

Animation Creator Reference

Animation Editor



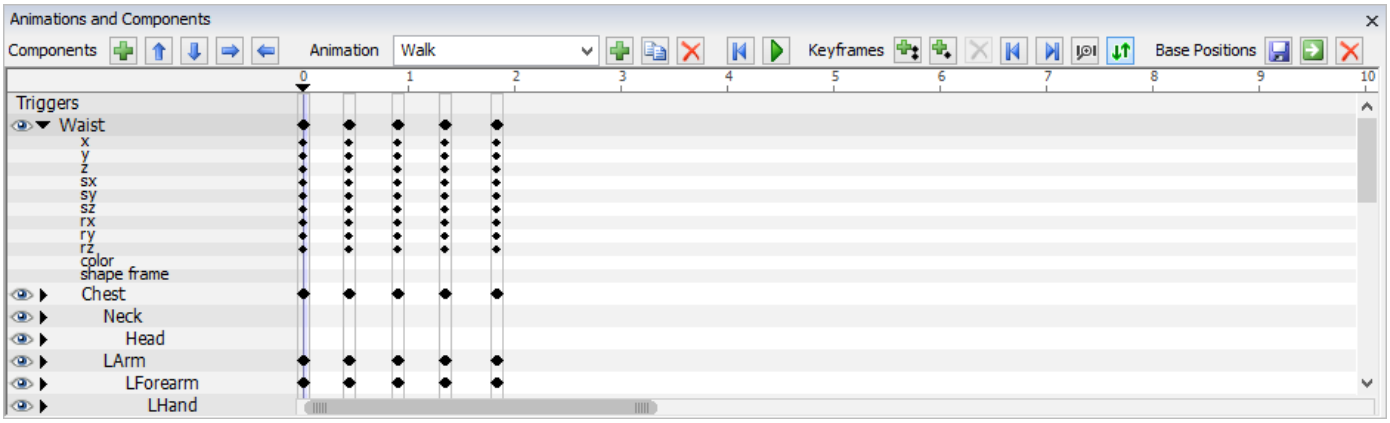
Click on a component in one of the 3D views to highlight that component in the other 3D views and in the Animations and Components timeline.

The 3D views work like the standard 3D view. You can move around in the views (though rotation is only allowed in the perspective view) and position, size and rotate components.

⛶- Hides the other 3D windows and expands the selected 3D window to take up the entire Animation Editor view.

Flip Axis - Available in the Side, Front and Top 3D Views through the right-click menu. This option flips the axis of the view. For example, if you flip the Axis that is currently display the Top view, it will now display the Bottom view.

Animations and Components Window



Components

- Adds a new component. Select a component from the list and hit the Delete key to remove the component.

- Removes the selected component up or down in the list.

- Removes the selected component left or right in the list. This creates parent/child relationships with the components. When a parent component's position or rotation is moved, all of its child components will also have their position/rotation moved with respect to their parent.

Animation

Animation combobox - This field displays the current animation being edit as well as a list of all animations for this object. Select an animation to display the animation in the timeline.

- Adds a new animation to the object.

- Duplicates the current animation.

- Removes the current animation from the object.

- Moves the current time cursor to the beginning of the animation (time 0).

- Plays/Stops the animation. Animations will automatically loop playback.

Keyframes

- Adds a new keyframe to the animation for all components.

- Adds a new keyframe to the animation for the selected component only.

- Removes the selected keyframe(s).

- Moves the time cursor to the previous or next keyframe

- Zooms the timeline in or out to make all keyframes fit on the screen.

- Sync the 3D Views. If toggled, the 3D views will update the object to display the current values of all its components as you move the time cursor in the timeline.

Base Positions

- Saves the current values of all the object's components as the base position. This is how the object will be displayed in the model when the model is reset.

- Updates the 3D views to display the object in it's currently saved base positions. This does not affect the timeline.

- Removes the saved base positions.

Timeline

Click on a keyframe to select it (turns red). You can drag keyframes around the timeline to reposition them.

You can zoom the timeline in and out by dragging the ends of the bottom scroll bar. You can also change the current time cursor by clicking and/or dragging anywhere on the timeline.

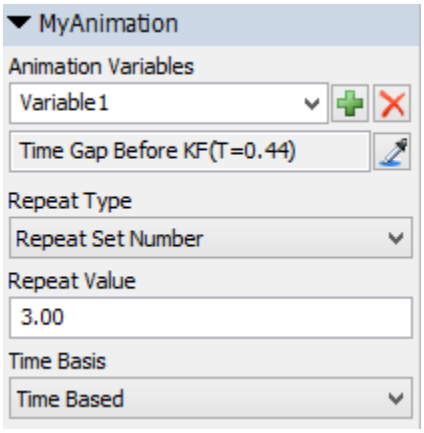
Select a component to highlight it in the Animation Editor's 3D views.

Triggers - Shows keyframe triggers for the animation.

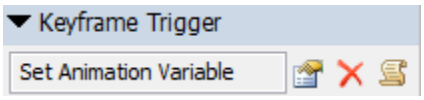
- Toggles the component as visible or hidden.

- Opens the individual properties of that component, displaying keyframes for the position, size, rotation etc.

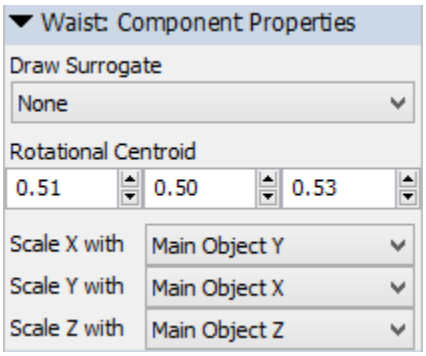
Quick Properties



See the Concepts page for more information on Repeat Types and Time Basis.



For more information on Keyframe Triggers, see the Concepts page.

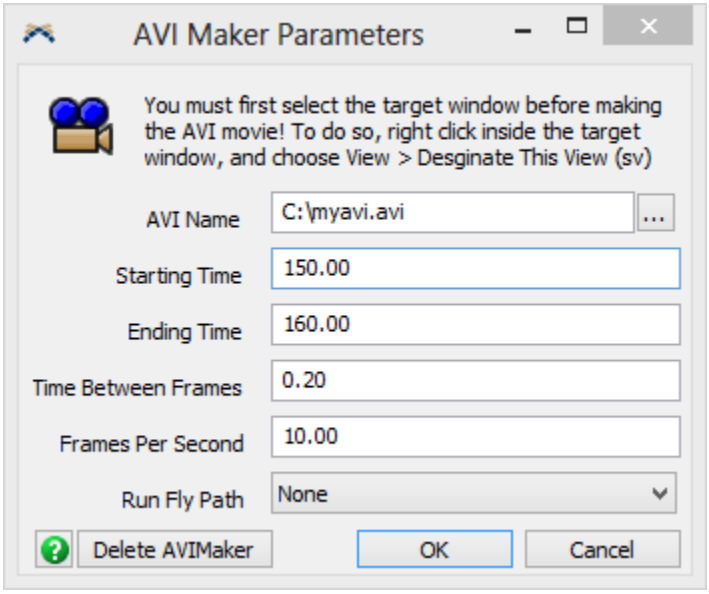


For more information on Surrogates, see the Concepts page.

AVI Maker

1. Concepts
2. Example

AVI Maker Concepts

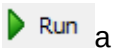


The Presentation Builder is accessed from the Tools menu > **Presentation** > **AVI Maker**.

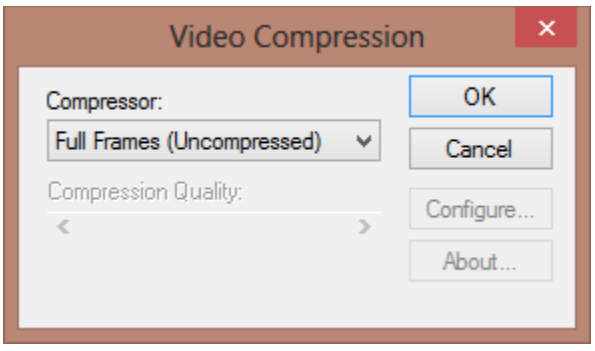
The AVIMaker is an object that is added to the Tools folder of the model tree when the AVI Maker option is selected in the Tools menu. It is a special object in the model that calls the commands to make an AVI file of the model while it is running. It will create an AVI file as long as the AVIMaker object is in the model. If the user does not want to make the AVI file, they need to delete the object from the model.

Before running the model, the user must designate a view to record. This is done by right-clicking on the desired 3D view selecting **View > Designate This View (sv)**. The model may run very slowly while the AVI file is being recorded. It will not respond to the speed slider bar in the run control window during that time.

Note on window size: The AVI Maker will capture the entire 3D window as it saves each frame. The larger the window, the larger the video file and the longer it will take to export the avi file.

Once you hit 

a window will appear allowing you to specify the type of compression you want the avi file to use.



Select the desired compression then press OK to close the window.

AVI Name - This is the name and path of the file that the AVIMaker will write to. It should have an .avi extension.

Starting Time - This is the simulation time that the AVIMaker should start recording the AVI file.

Ending Time - This is the simulation time that the AVIMaker will stop recording the AVI file. It is recommended that the model not be stopped before this time, as it may corrupt the file being written to.

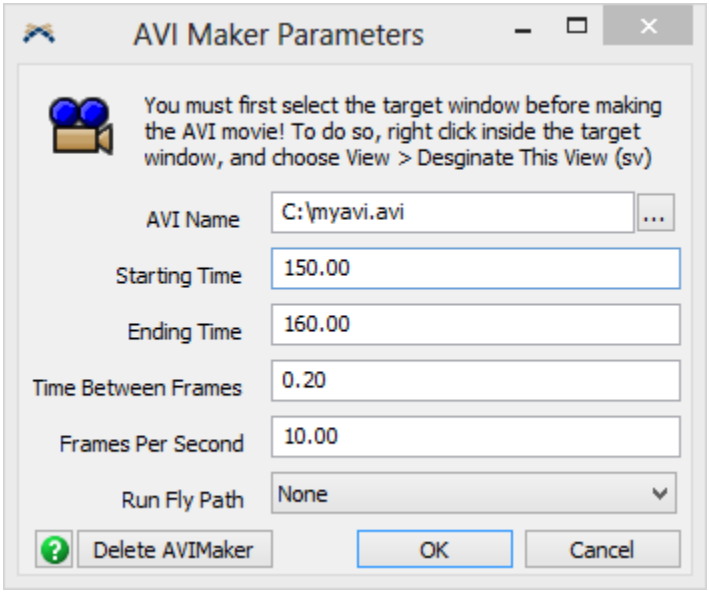
Time Between Frames - This is how much simulation time passes in the model between recorded frames.

Frames Per Second - This number defines how many frames per second the AVI file plays back at. This value is based on actual seconds not simulation time.

Run Fly Path - If you have created a fly path in your model you can select that fly path in this drop down box. The view will follow that fly path as the model runs, giving more interest to the recorded avi.

Delete AVIMaker - When this button is pressed, the AVIMaker is deleted from the model. The model will run at normal speed again, and the AVI file will not be generated.

AVI Maker Example

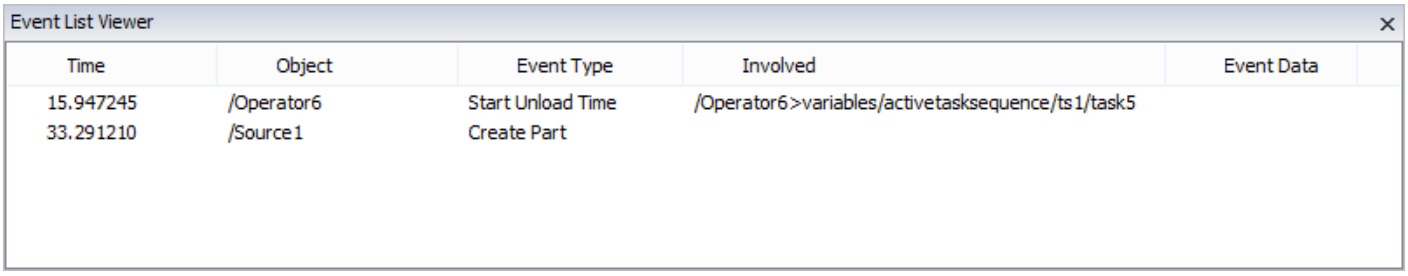


Properly working with the AVI Maker

The AVI maker can often be tricky to work with. Here are steps to go through to make sure that your AVI is created with as little hassle as possible.

1. Open the AVI Maker window (adding an AVIMaker object to your model) by going to the Tools menu > Presentation > AVI Maker.
2. Fill in the above mentioned fields appropriately.
    a. Make sure the name of the avi file is not the name of a file that already exists, so as to not overwrite a file.
    b. Set the start and stop times according to the simulation time that you want the avi to be recorded during.
    c. Set the frames per second value to whatever you want your avi playback speed to be. 10 frames per second is usually reasonable.
    d. Set the time between frames value based on the frames per second value you specified. Find an ideal run speed that you want the avi to record for the model (from the simulation run control panel). The time between frames should be calculated as that ideal run speed divided by the frames per second
3. Right click in the 3D view you want to be recorded and select View > Designate This View (sv).
4. Resize the 3D view to the resolution you want for the avi movie. Setting a smaller window size can dramatically increase the speed that the avi maker can create the movie.
5. Reset and run the model
6. A window will pop up about the codec to use. Enter the codec/compression that you want to use.
7. IMPORTANT: Wait until the avi maker is finished. Once the model gets to the avi's starting time, do not press any buttons or click on anything until the model time is past the ending time you have specified for the avi maker.
8. Once the model has run past the avi maker's ending time, stop the model. Do not hit reset again until you have deleted the AVI Maker.
9. Delete the avi maker using the Delete AVIMaker button.

Event List



The Event List is accessed from the Debug menu > **Event List**.

The Event List shows all the pending events for the model. It is useful for seeing when different events will occur in order to debug modeling issues. If you have a problem that happens during a particular event, the Event List is useful for seeing information about that event to help track down the source of the problem.

If you want to only view the events for a particular object, you can right-click on the object in the 3D View and select **View > View Object Events**.

Time - This is the time that the event occurs.

Object - This is the path to the object, relative to the model, that the event affects.

Event Type - This is the type of event. It is the event code and will show a number value for event codes without registered names. You can use the "seteventlistlegendentry" application command to register a name for custom event types.
For example: applicationcommand("seteventlistlegendentry", 102, "My Event Type", 0); will set event code 102 to show "My Event Type" as its name in the list.

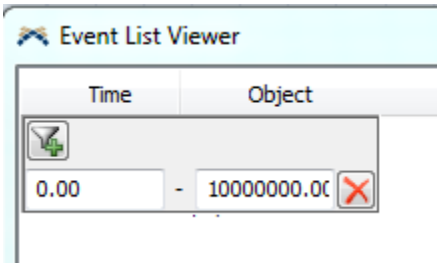
Involved - This is the path to the involved object for the event.

Event Data - This value's use depends on the event and may not be used for all event types.

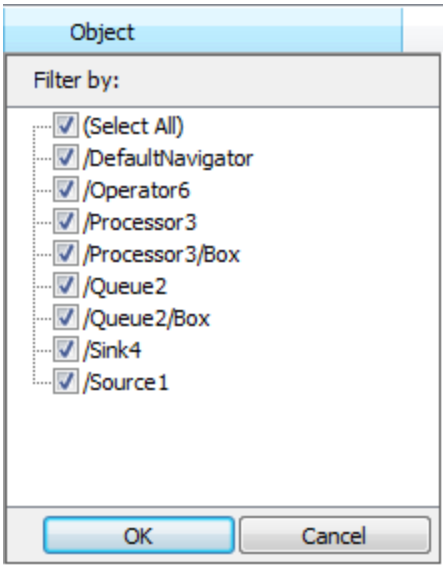
Filters

The Event List can be filtered based on the Time, Object and Involved columns. Columns with an active filter will display a . To add/edit a filter, left-click on the header name for the desired filter.

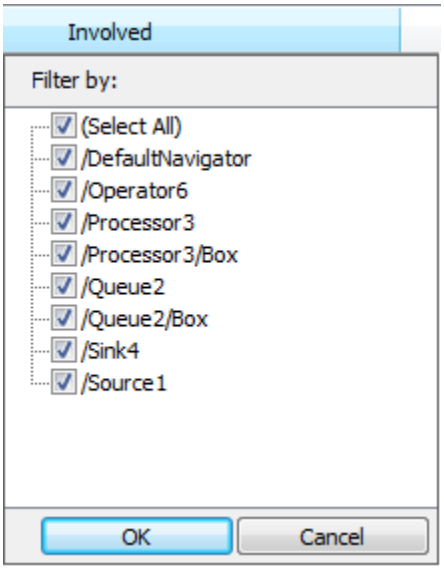
Time - Each time filter has a begin (left) and an end (right) field. Only events that occurred within those two times will be displayed. To add a filter, click the icon. To remove a filter, click the icon.



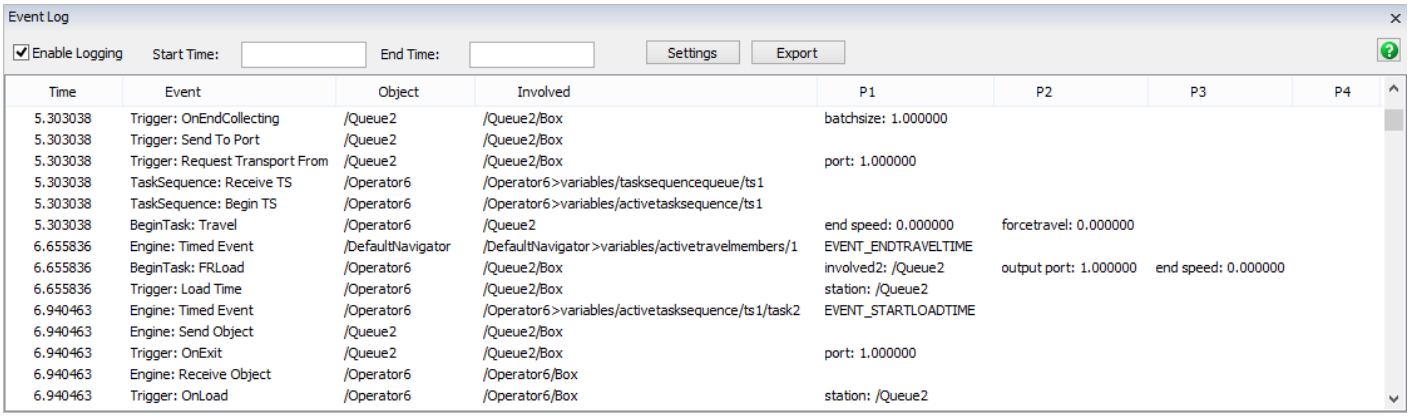
Object - This list allows you to filter the event list by which object generated the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Involved - This list allows you to filter the event list by which object is involved in the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Event Log


Event Log
Enable Logging
Start Time:
End Time:
Settings
Export
Time
Event
Object
Involved
P1
P2
P3
P4
5.303038 Trigger: OnEndCollecting /Queue2 /Queue2/Box batchsize: 1.000000
5.303038 Trigger: Send To Port /Queue2 /Queue2/Box
5.303038 Trigger: Request Transport From /Queue2 /Queue2/Box port: 1.000000
5.303038 TaskSequence: Receive TS /Operator6 /Operator6>variables/tasksequencequeue/ts1
5.303038 TaskSequence: Begin TS /Operator6 /Operator6>variables/activetasksequence/ts1
5.303038 BeginTask: Travel /Operator6 /Queue2 end speed: 0.000000 forcetravel: 0.000000
6.655836 Engine: Timed Event /DefaultNavigator /DefaultNavigator>variables/activetravelmembers/1 EVENT_ENDTRAVELTIME
6.655836 BeginTask: FRLoad /Operator6 /Queue2/Box involved2: /Queue2 output port: 1.000000 end speed: 0.000000
6.655836 Trigger: Load Time /Operator6 /Queue2/Box station: /Queue2
6.940463 Engine: Timed Event /Operator6 /Operator6>variables/activetasksequence/ts1/task2 EVENT_STARTLOADTIME
6.940463 Engine: Send Object /Queue2 /Queue2/Box
6.940463 Trigger: OnExit /Queue2 /Queue2/Box port: 1.000000
6.940463 Engine: Receive Object /Operator6 /Operator6/Box
6.940463 Trigger: OnLoad /Operator6 /Operator6/Box station: /Queue2

The Event Log is accessed from the Debug menu > **Event Log**.

When **Enable Loggin** is checked, the Event Log will create a record of events that occur in the model. It is useful for seeing the order in which certain events took place. For each event that happens in the model, multiple entries may be made in the Event Log to explain what happened during that event. These multiple entries will all have the same time and all be logged simultaneously when you press the Step button. The event log will be cleared when the model is reset.

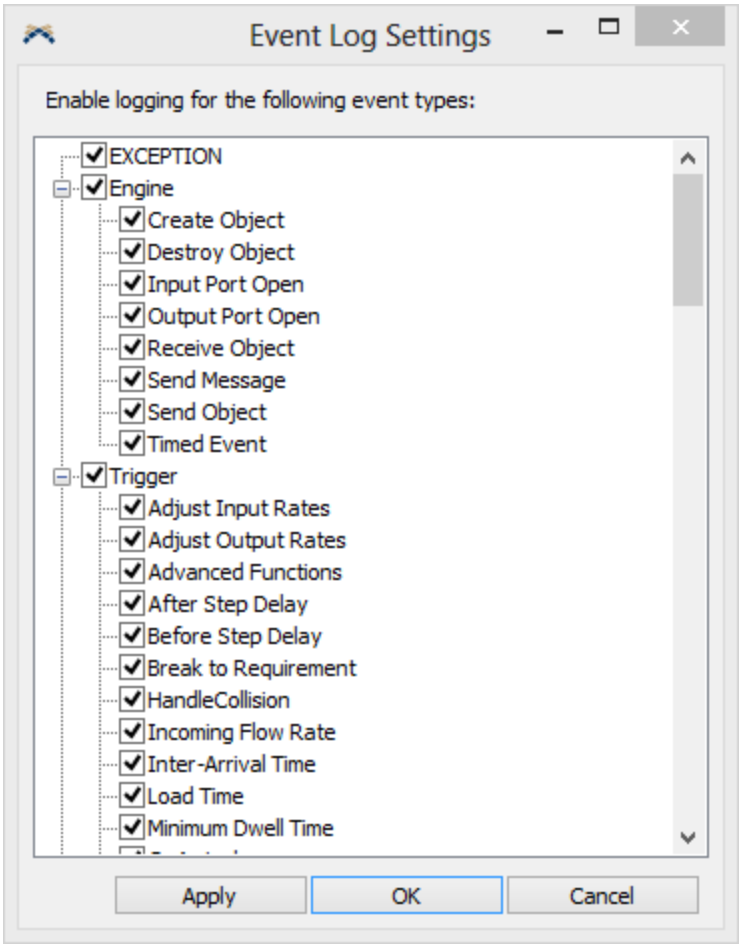
Some exceptions will be recorded in the event log. The entry immediately preceeding the exception entry will give you a clue as to where the code is that caused the exception to happen. This is particularly useful if the exception was caused by improper code in an object's trigger. More information about the exception may be available in the System Console. The model may not be behaving correctly if there are exceptions happening in the code.

Enable Logging - This will enable or disable event logging. The model will run much slower when logging is enabled so you should disable logging when you are finished using the event log.

Start Time - If you only want to log a specific time period, you can enter a start time for when the logging will begin. This will automatically be applied after editing this field without having to reset and rerun the model.

End Time - Optionally, you can specify an end time for when you want the logging to stop. If the end time is less than or equal to the start time, it will be ignored.

Settings - Within the settings window, you can set up which events should be recorded in the event log. Events that have already been recorded will not be affected by changing these settings. Events that occur after changing these settings will only be recorded if they are enabled here.



Export - This will export the Event Log as a csv file. If will only export valid events, ignoring any events that have been filtered out.

The Table

Time - This is the time that the event happened. The entries happened in order from top to bottom. Entries recorded with the same time happened in the order shown and may have happened during the same model event.

Event - This is the type of event. You can enable or disable logging for certain event types in the Settings window.

Object - This is the path to the event's object.

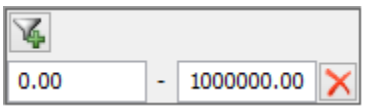
Involved - This is the path to the involved object for the event.

P1 - P4 - These values depend on the event and may not be used for all event types. Usually they give you information about what parameters were passed into the event or more information about the event type. This is useful for debugging if parameter values are not what you expect them to be.

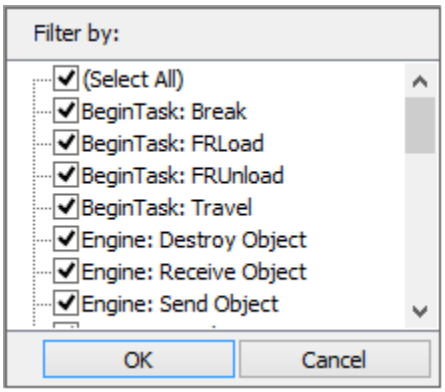
Filters

The Event Log can be filtered based on the Time, Event, Object and Involved columns. Columns with an active filter will display a . To add/edit a filter, left-click on the header name for the desired filter. Event log entries that are no longer displayed because they have been filtered will not be exported with the Export button.

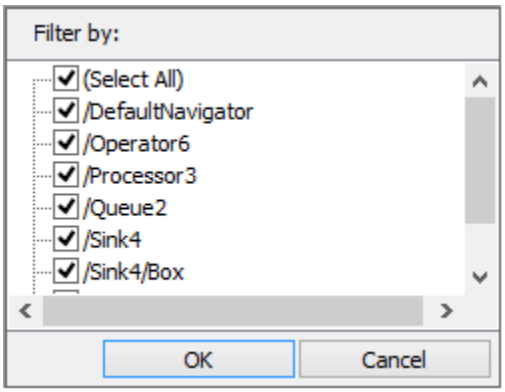
Time - Each time filter has a begin (left) and an end (right) field. Only events that occurred within those two times will be displayed. To add a filter, click the icon. To remove a filter, click the icon.



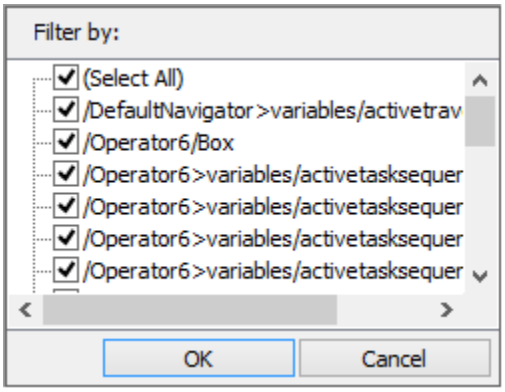
Event - This list allows you to filter the list by which event or trigger the event was associated with. To include or exclude an event from the list, check or uncheck the box next to its name.



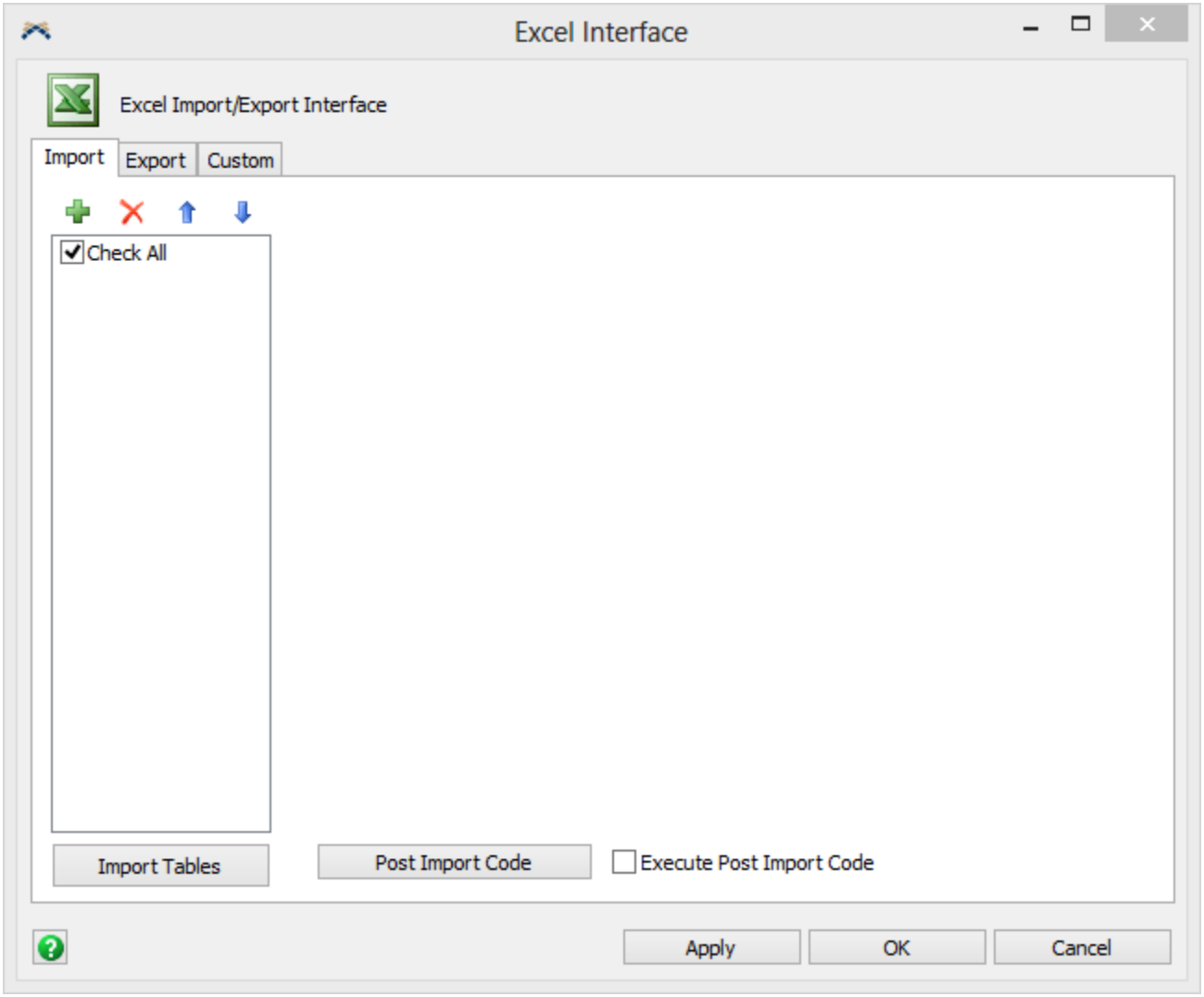
Object - This list allows you to filter the list by which object generated the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Involved - This list allows you to filter the list by which object is involved in the event. To include or exclude an object in the list, check or uncheck the box next to its name.



Excel Interface



The Excel Interface was designed to make importing and exporting multiple worksheets from more than one workbook very fast and easy to do. You can also create your own custom Import/Export code.

Overview

Multiple Table Excel Import (MTEI)

The MTEI is capable of automating much of the import process in terms of the table size and cell data type. If you allow the MTEI to be more automated in its implementation, it is extremely useful for importing data that will change over time.

Multiple Table Excel Export (MTEE)

The MTEE allows you to export multiple tables to multiple different Excel workbooks. The controls and features of this page are the same as the MTEI page. See the documentation above for more information on this page.

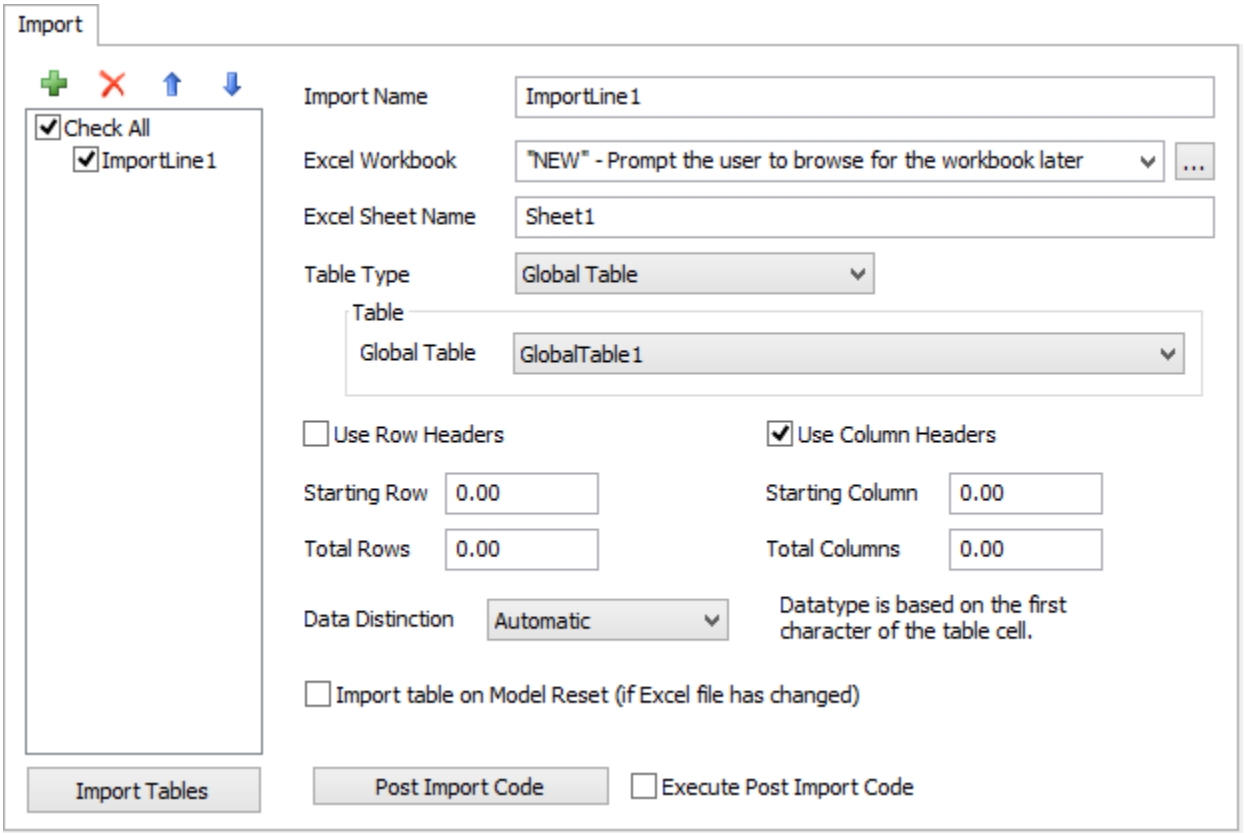
Custom Import/Export

The custom Import/Export page allows you to write your own custom code to import and export to Excel workbooks. There is sample code in the picklists on importing and exporting from Excel.

Excel Interface Pages

- **Import Page**
- **Export Page**
- **Custom Page**

Import Page



Import Lines (list)- Displays all of the import lines for the MTEI. You can use the Add and Remove buttons to add additional lines. You may also rearrange lines to group import lines that are importing from the same Excel workbook (this will improve speeds greatly as opposed to trying to switch back and forth between the same workbooks).

Import Tables - The MTEI will go in order (from the top) through each checked line of the Import Lines window and execute the import based on its properties specified on the right side.

Note: The MTEI and MTEE may also be started by calling **excelmultitableimport()** or **excelmultitableexport()** respectively in a flexscript node.

Import Name - The import name is only used in the Import Lines window to help identify the line.

Excel Workbook

The Excel Workbook is where you define the name of the Excel workbook file that contains the information that you want to import. There are four ways to enter information into this column.

Unknown workbook name or location ("NEW")

If the name or location of the workbook that you want to use are unknown or will change over time then you can select the "NEW" option. Using "NEW" in this field will cause the browse window to open, prompting the user to find the Excel file they want to use when executing the import. This is an extremely useful option when the input data will change with different runs or users.

Same as the previous location (Blank)

If you want to continue to use the same workbook as the previous import line then you can leave this field blank. This is the recommended option when you are importing information from multiple sheets in the same workbook. You can reorder import lines using the arrows on the left side of the window in order to group imports that use the same workbook.

Absolute Path (ABSOLUTE)

If the location and the name of the workbook will not change for the entire use of the model then you can enter the absolute path of the Excel workbook. For example "C:\tempdirectory\myfile.xls"

If the file is unable to be found, the import will be stopped and you'll have the option to skip the file or manually select a file to import

Note: The ".xls" extension is essential in order for FlexSim to find the right Excel spreadsheets. You can also use ".xlsx" to import Excel 2007 or newer spreadsheets.

Relative Path (RELATIVE)

If your Excel file is in the same directory as your model, or in a sub directory, you can enter the relative path of the Excel file, or browsing for the file will give the relative path to the file. Alternitvely, you can enter the relative path with respect to the install directory of FlexSim. For example "userprojects\myproject\myfile.xls"

If the file is unable to be found, the import will be stopped and you will have the option to skip the file or manually select a file to import.

Excel Sheet Name - The name of the Excel sheet that contains the import information. For example "Sheet1". If the MTEI does not find the name of the sheet because it does not exist in the workbook or has been entered incorrectly, the import will pause and alert you of the problem. You will then be given the option to exit the import completely or skip the offending import row and continue with the next one. Hint: Look for spaces at the beginning and end of the name if you are alerted that a sheet name does not exist.

Table Type - Select the type of table in FlexSim that should receive the data from the import. There are 8 table types:

1. Combiner Component Table
2. Conveyor Layout Table
3. Global Table
4. Pipe Layout Table
5. Source Table
6. Time Table
7. Traffic Control Table
8. Other

Headers - Implementing headers will cause the MTEI to import the column and or row names for the table. This is useful for helping you to identify the columns and rows later in FlexSim. The row or column for the header information is automatically calculated. The header information should always come before any data distinction information or actual data.

Starting Row and Starting Column - The starting row and starting column fields determine where the MTEI will look on the Excel sheet for the data it needs to import. Enter the starting location for your data in these cells not the location of the headers or data distinction information. If you leave the values for these cells at 0, the MTEI will automatically adjust where it imports the data from. If you always leave your data at the top left of the worksheet you will never need to enter a value other than 0 in these cells regardless of whether or not you have headers or data distinction information in front of the data.

Total Rows and Total Columns - The Total Rows and Total Columns fields determine the amount of rows and columns that the MTEI will import. If you set these entries to 0 the MTEI will automatically calculate the number of rows or columns for you by searching for the first empty cell in a chosen "search" row or column. The "search" row or column will be based on the header or data distinction row or column if they are

available. If there is no header or data distinction row or column, then the "search" row or column will be the first row or column of data. Letting the MTEI calculate the number of rows or columns for you is a great way to allow the developer and or user of the model to add or delete rows and/or columns from the table as necessary without having to worry about changing any other values.

Note: "Searching" requires more processing. With large tables, this will slow down import time.

Note on automatic resizing: The MTEI automatically sizes the FlexSim table that it is importing into to fit the size of the table that it is importing.

Data Distinction - Data distinction refers to the way in which the MTEI will interpret the incoming data and the way that it will format the FlexSim table. The data distinction allows you to import tables with both numeric and text data.

1. Numeric - all data is assumed to be numeric
2. Automatic - data distinction is determined by whether the cell contains a number or string data
3. Row - data is defined by the row above the first row of data in Excel
4. Column - data is defined by the column before the first column of data in Excel
5. String - all data is assumed to be strings

Automatic Data Distinction - When using automatic data distinction, the excel importer will go through each cell and determine whether the cell contains number data, or string data. Each value is initially read in as a string, and then all characters in the string are checked. If the string contains any non-number characters, the data will be imported as a string. In Excel, you have format options. Not all format options may read in properly using this data distinction. For example, the fraction format will read in as a string when using automatic data distinction. If a cell comes in improperly, change the formatting or use the numeric data distinction. Empty cells will be read in as a string.

For options 2 and 3,the first cell in a row or column becomes the data distinction cell. For example: if the cell contains a number value, all cells in that row or column will be imported as number data. The data distinction row or column should always be placed before the actual data but after any header information. The values that can be entered in the data distinction row or column in Excel are listed below:

1. Numeric Data
2. Text Data
3. Flexscript Data
4. C++ Data

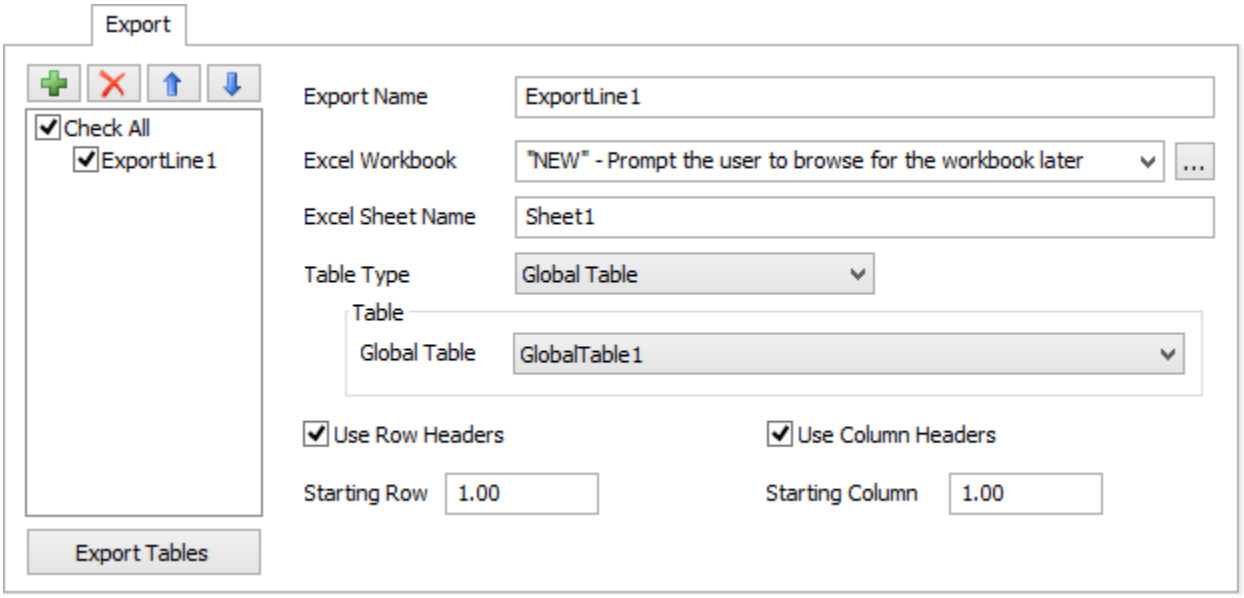
The MTEI will automatically format the node in the table to be numeric or text and to be built as Flexscript or C++ depending on the data distinction. If the table imports C++ data the MTEI will prompt the user to recompile the model after the import is finished.

Import Table on Model Reset - If checked, FlexSim will re-import this table when the model is reset. This will ONLY occur if the Excel file has been changed since the last time the table was imported.

Post Import Code - After all the import lines are executed, the MTEI can execute Post Import Code. You can write custom code in this trigger to do any additional operations after all tables have been imported.

Execute Post Import Code - If checked, the Post Import Code will be executed once all tables have been imported.

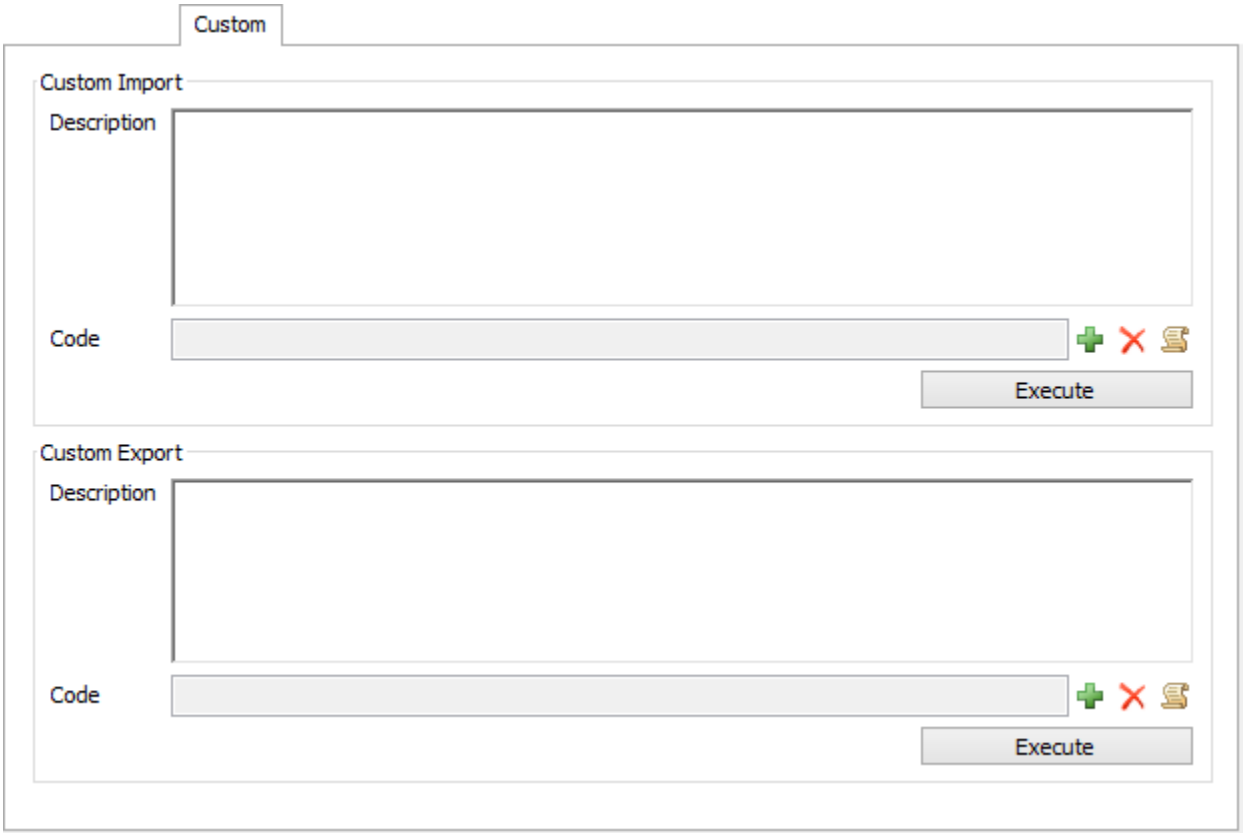
Export Page



Unlike the MTEI, the MTEE does not use any data distinction when writing to the Excel file. The data type is taken from the FlexSim node data type.

There is also no Number of Rows or Number of Columns fields. The export will take the entire table starting at the specified row/column.

Custom Page



Description - This field has no impact on the custom import/export. It is purely for the user's information.

Code - Enter your own code to import/export from Excel.

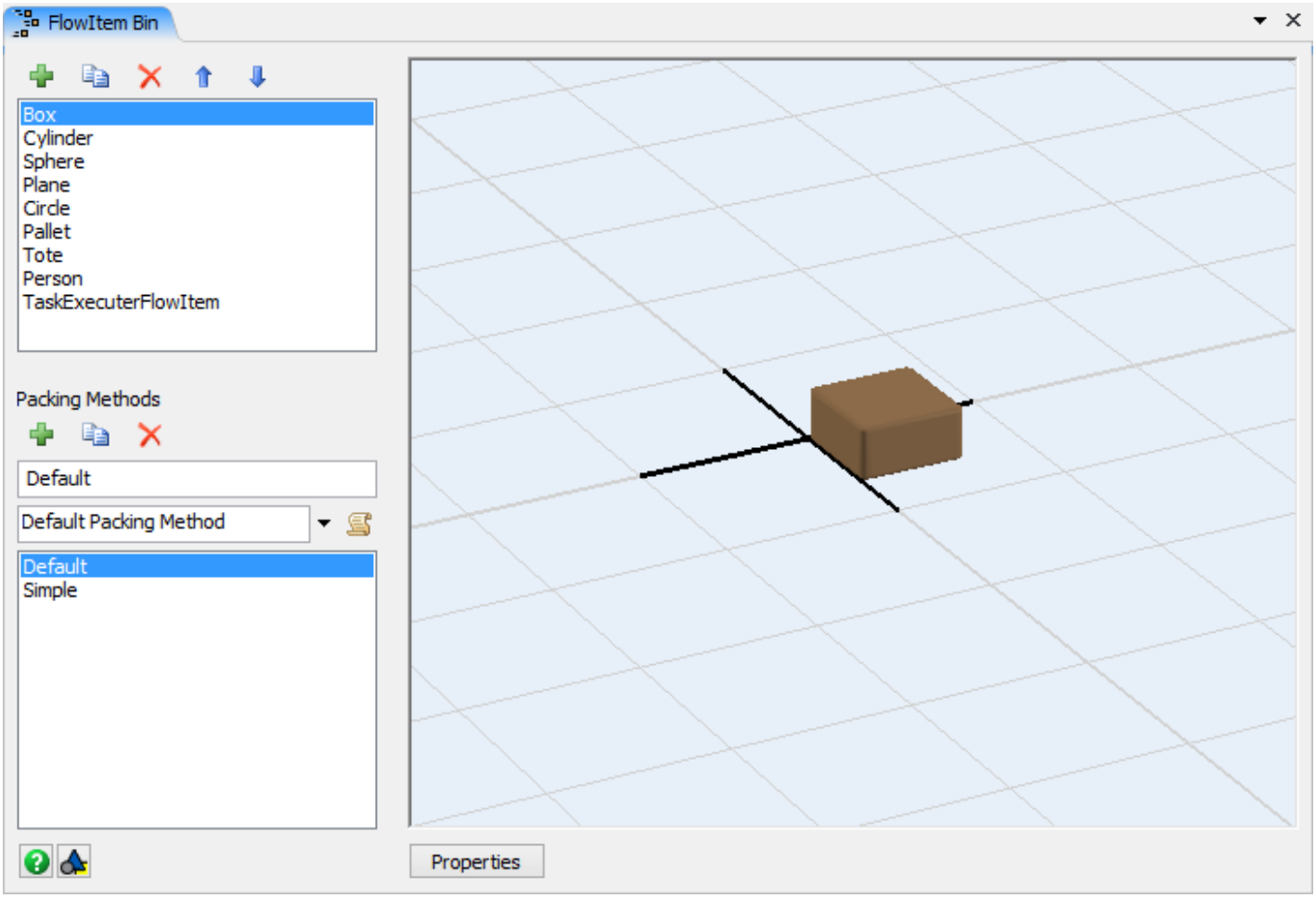
To execute your custom import or export code through a flexscript node or trigger, call the following:

```
treenode excelauto = node("/Tools/ExcelAuto", model());

executefsnode(getvarnode(excelauto, "CustomImport"),NULL);

or

executefsnode(getvarnode(excelauto, "CustomExport"),NULL);
```

Flowitem Bin

1. Concepts
2. Reference

Flowitem Bin Concepts



You can get to the Flowitem bin through the Tools menu, or through FlexSim's toolbar.

The Flowitem bin stores all of the Flowitems used in your model. You can learn more about Flowitems in the FlexSim Concepts - Flowitems page. Different classes of flowitems are created in this window and are stored in the Flowitem Bin.

Flowitems that are created in the model are exact copies of the Flowitems in the Flowitem Bin. A Source object specifies what Flowitem class to create. It then creates a copy of the Flowitem and places it in the model.

Flowitem Types

There are three types of flowitems, Basic, Container and TaskExecuter. When a new model is created, a default set of Flowitems is added to the Flowitem bin. The Box, Cylinder, Sphere, Plane, Circle and Person are all Basic Flowitems. The Pallet and Tote are Container Flowitems and the TaskExecuterFlowItem is a TaskExecuter Flowitem.

Basic Flowitems

Basic Flowitems have the following properties:

Name - Each Flowitem has its own name. However, unlike FixedResource and TaskExecuter objects, Flowitem names do not need to be unique within the model. If the Flowitem is named "Box" in the Flowitem bin, then all copies of that Flowitem will be named "Box" unless the name is explicitly changed.

Itemtype - See the FlexSim Concepts - Itemtype page for more information on Itemtypes.

Labels - See the FlexSim Concepts - Labels page for more information on Labels.

Location, Size, Rotation - Flowitems take up physical space in your model. This allows you to set your Flowitems to be the same size as the parts you are modeling (ie bottles in a bottling line).

Color - As with other objects in FlexSim, Flowitems have a designated color. That color can be changed in the Flowitem bin, but it can also be changed in the model as the Flowitem makes its way through processes. Often, colors are used to represent specific product types and allow the modeler to more easily follow what is happening in a model.

3D Shape - See the 3D Media section for more information on 3D Shapes.

Shape Frames - As with other objects, Flowitems can have Shape Frames. This is particularly useful with Flowitems as it can help show how a part changes as it moves through the model. For example, in a bottling line, the Flowitem might begin as a piece of plastic or glass, then become a bottle, then it is filled, and a cap is put on top. This could be shown by creating five different 3D shapes and changing the Shape Frame.

Animations - New with FlexSim 7, Flowitems can have their own custom Animations using the Animation Creator.

Container Flowitems

Container Flowitems have all of the same functionality as Basic Flowitems. Though all Flowitems can act as containers, only the Container Flowitem executes code when another item is placed inside of it (packed). You can create custom packing methods in the Packing Methods section of the Flowitem bin.

Container Flowitems have an extra page in their properties window that allows you to specify which Packing Method to use. For more information, see the Container Page.

TaskExecuter Flowitems

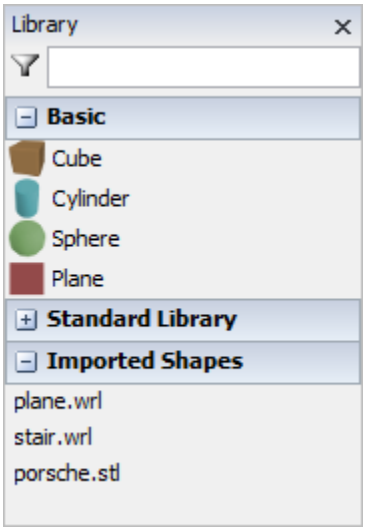
TaskExecuter Flowitems have all of the same functionality as Basic Flowitems. They also have the ability to act as a standard TaskExecuter object. Unlike other Flowitems, they can have their own connections which can be added dynamically using the *contextdragconnection* command.

TaskExecuter Flowitems have the following pages added to their Properties window:

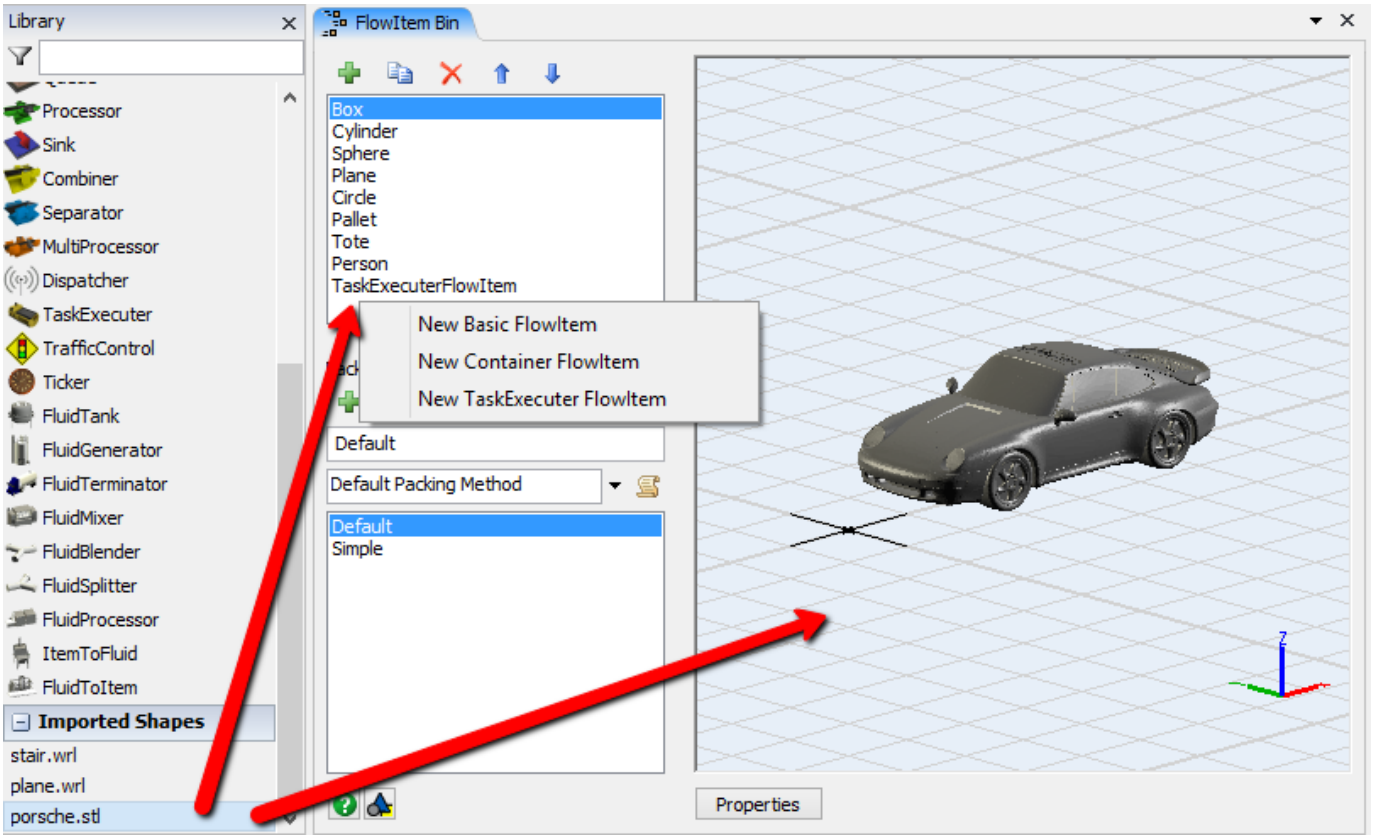
- TaskExecuter
- Breaks
- Collision
- Triggers
- Statistics

Library Icon Grid

When the Flowitem Bin is active, the Library Icon Grid changes to display a list of shapes:



Shapes that have been imported into your model will appear at the bottom of the Library. You can drag and drop shapes from the Library onto the 3D view to quickly change the 3D shape of the Flowitem, or you can drag and drop into the Flowitem Class list to create a new Flowitem:

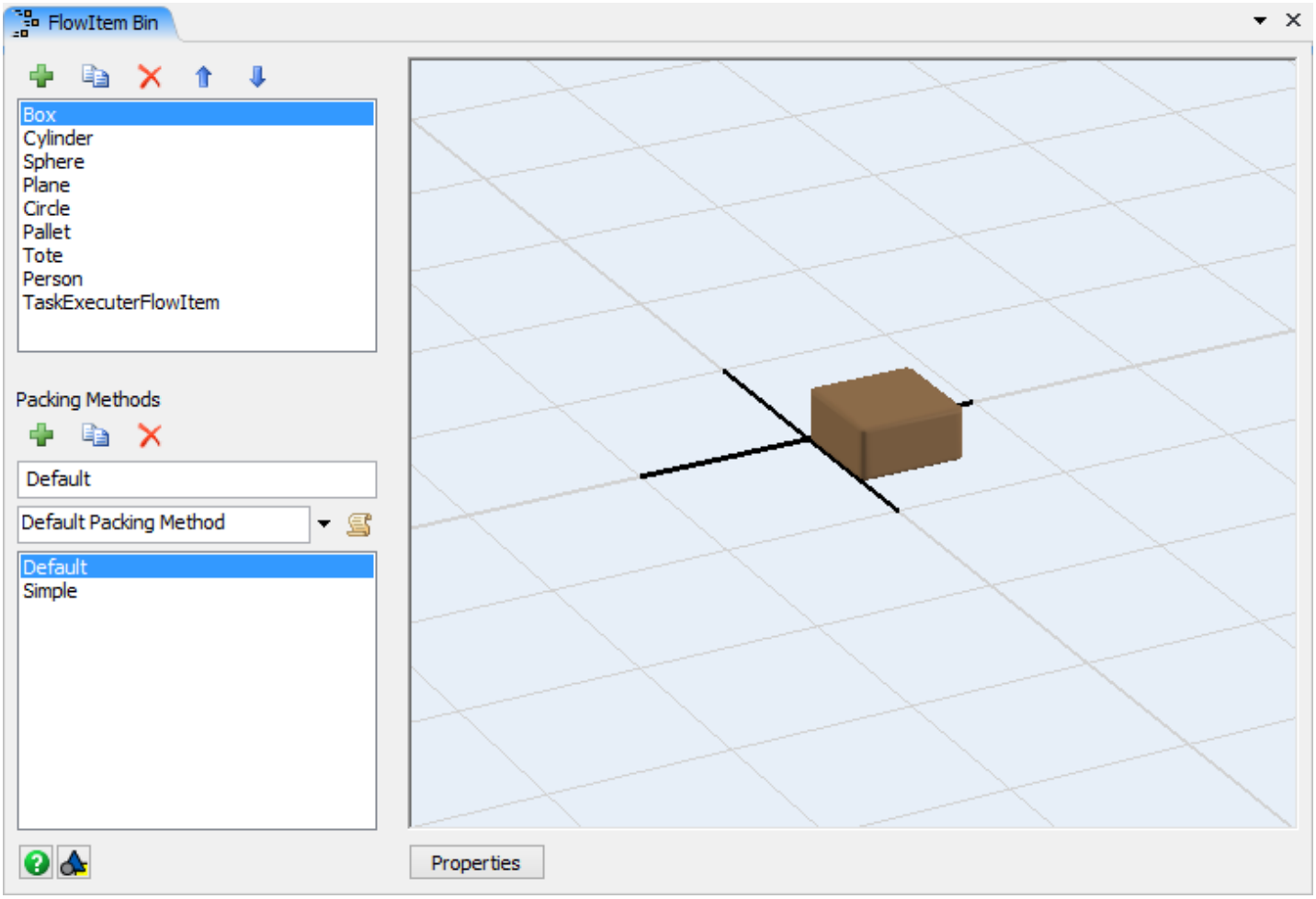


You can filter the list of shapes by entering text into the field.

Flowitem Bin Reference

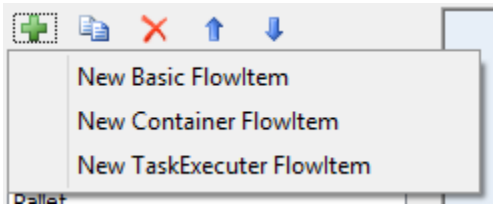
Topics

- **Flowitem Classes**
- **Packing Methods**
- **Right-Click Menu**
- **Quick Properties**



Flowitem Classes

- Adds a new Flowitem of either Basic, Container or TaskExecuter type to the Flowitem Bin.



- Duplicates the currently selected Flowitem.

- Removes the currently selected Flowitem from the Flowitem Bin.

- Reorders the currently selected Flowitem up or down in the list.

Flowitem Class List - This list contains all of the available Flowitem classes. When one is selected, it is shown in the 3D view to the right. The Quick Properties will also update to display the properties for that Flowitem class.

Packing Methods

- Adds a new Packing Method.

- Duplicates the currently selected Packing Method.

- Removes the currently selected Packing Method.

Packing Method Name - Change the name of the Packing Method by entering the new name in this field.

Packing Method Picklist - Allows you to pick from a list of preset packing methods or to edit the code manual in a Code Editor window.

Packing Methods List - Displays the list of all Packing Methods.

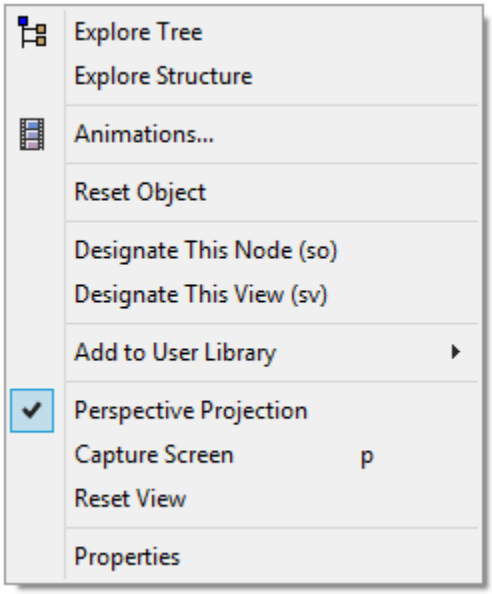
Assigning Packing Methods: To assign a packing method to a Container Flowitem, open the Flowitem's properties window and set the Pack Contents of the Container Page.

- Displays the help page.

- Adds the selected Flowitem to a User Library as either a Draggable Icon or an Auto-Install Component.

Properties - This button opens the currently selected Flowitem's Properties window. You can also access this window by double-clicking on the Flowitem in the 3D view.

Right-Click Menu



Explore Tree - Opens a Tree Window and displays the Flowitem object in the Tree.

Explore Structure - This option brings up a tree window exploring the tree structure of the 3D window itself.

Animations... - Opens the Animation Creator to edit the object's animations.

Reset Object - This resets the x/y/z rotation and the z location of the object to 0.

Desginate This Node (so) - This designates the object as the "selected object", which can then be referenced in code as so(). You will usually use this option for writing code in the script console. There can only be one so() at any time.

Designate This View (sv) - This designates the window as the "selected view", which can then be referenced in code as sv(). You will usually use this option for writing code in the script console. There can only be one sv() at any time.

Add To User Library - Adds the Flowitem to a User Library as either a Draggable Icon or an Auto-Install Component.

Perspective Projection - Toggles the 3D view from Orthographic to Perspective view. For more information see the Perspective View page.

Capture Screen - Saves a PNG file of the current 3D view.

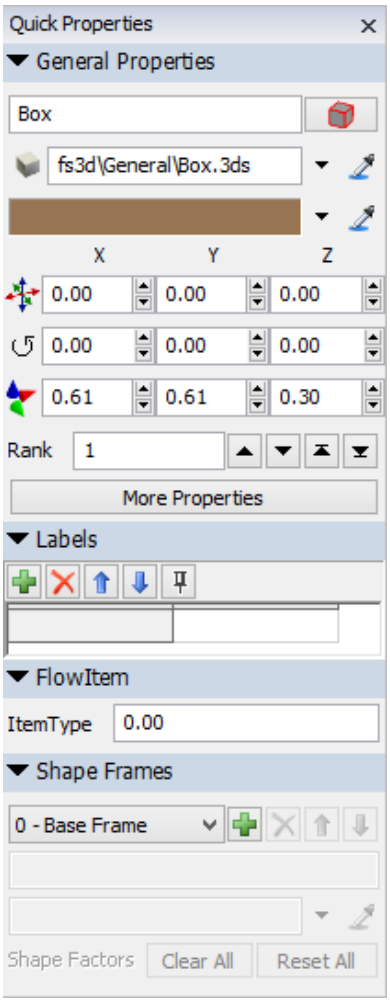
Reset View - Resets the view rotation and position to 0.

Properties - Opens the Flowitem's Properties window.

3D View - Displays the Flowitem in 3D.

Quick Properties

The Quick Properties will change to display the following panels:



The same properties can also be changed by opening the Flowitem's Properties window.

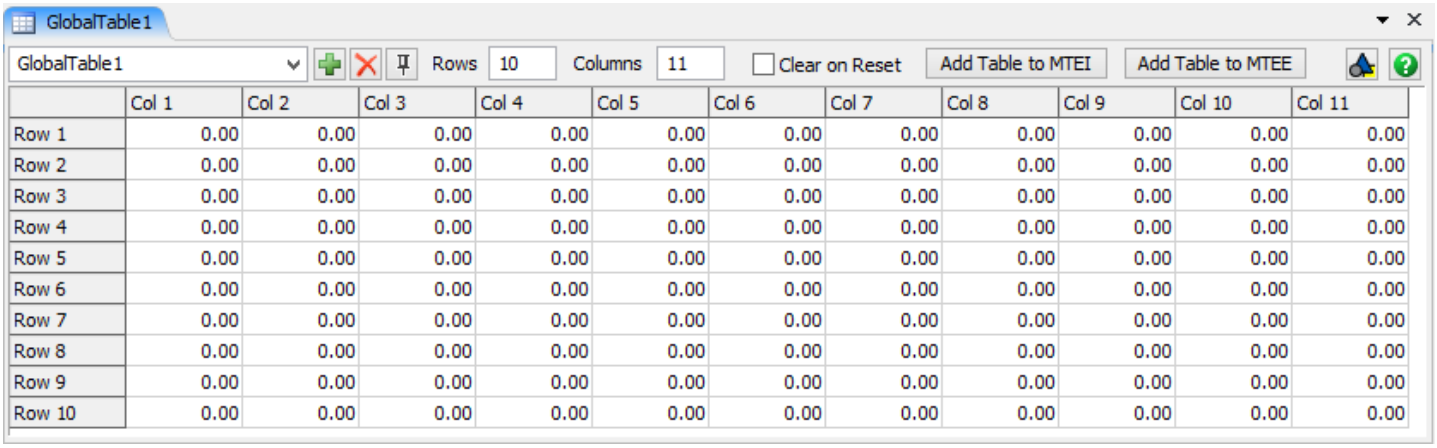
General Properties - See the General page for more information.

Labels - Add, remove and edit labels for this Flowitem.

Flowitem - Specifies the itemtype of the Flowitem. For more information see the FlexSim Concepts - Itemtype page.

Shape Frames - See the Shape Frames page for more information.

Global Tables



	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11
Row 1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Row 10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Global Tables are accessed from the Tools menu > **Global Tables**. These objects are not dragged out into the model. They are created through the Tools menu or through the Global Table window.

Global Tables can store numerical or string data. This data can be accessed by any object in the model using the various table commands. A model may have any number of Global Tables.

Editing the Table

To edit a cell in the table, click the desired cell and begin typing to overwrite all data in the cell, or double click on the cell to select the cell's contents. Use the arrow keys to navigate between cells. Cells hold number data by default, but can be set to hold string data by right-clicking on the cell and selecting Assign String Data. The right-click menu also has opens for insert/deleting rows and columns, clearing cell data, and sorting by column.

Name Combobox - This is the table's name and has a list of all of the model's Global Tables. The name should be memorable and describe the table's function. The commands to read and write to them access them by name. You can view other Global Tables in this window by clicking the dropdown arrow next to the name.

- Adds a Global Table to your model.

- Removes the current Global Table from your model.

- Pins the entire global table to a Dashboard as either a table of values, bar chart or line graph.

Rows - This is the number of rows in the table.

Columns - This is the number of columns in the table.

Clear on Reset - If this box is checked, all number cells in the table will be set to 0 and cells with string data will be cleared when the model is reset.

Add Table to MTEI - This buttons adds the table as a row in the Multiple Table Excel Import.

Add Table to MTEE - This buttons adds the table as a row in the Multiple Table Excel Export.

- This button lets you add this table to a user library as either a draggable icon or as a component for automatic install. For more information, refer to the user library documentation.

Note: A similar window to this is used when editing a label table from an object's labels or when editing a node as a table (accessible through the right-click menu in the Tree Window or Labels page). However, the , , Clear on Reset and buttons will not be available.

Commands

The following commands can be used with Global Tables. When specifying which Global Table, you may pass in the name of the Global Table as a string or you can use the reftable() command to pass in a double

(this method is faster than accessing the table by name for multiple table commands). These commands may also be used with other tables in FlexSim by passing in the table's node. See the commands documentation for more information.

reftable("GlobalTable") Returns a reference to the Global Table that can then be passed into subsequent table commands.

gettablenum(table, row, col) Returns the number value stored in the table at the cell specified by row and col.

gettablestr(table, row, col) Returns the string value stored in the table at the cell specified by row and col.

settablenum(table, row, col) Sets the number value in the table of the cell specified by row and col.

settablestr(table, row, col) Sets the string value in the table of the cell specified by row and col.

gettablecell(table, row, col) Returns a reference to the cell (node) in the table specified by row and col.

settablesize(table, row, col) Sets the size of table to rows and cols. See commands documentation.

gettablecols(table) Returns the number of columns in table.

gettablerows(table) Returns the number of rows in table.

clearglobaltable(table) Writes zeros to all the number cells and clears the text in all string cells of the table.

addtablecol(table) Adds a new column to the table.

addtablerow(table) Adds a new row to the table.

deletetablecol(table, col) Deletes the specified column from the table.

deletetablerow(table, row) Deletes the specified row from the table.

movetablecol(table, col, newcol) Moves the table column col to newcol.

movetablerow(table, row, newrow) Moves the table row row to newrow.

gettableheader(table, row/col, rowcolnr) See commands documentation.

executetablecell(table, row, col) Executes the cell in table as Flexscript. If the cell contains number data, returns the number.

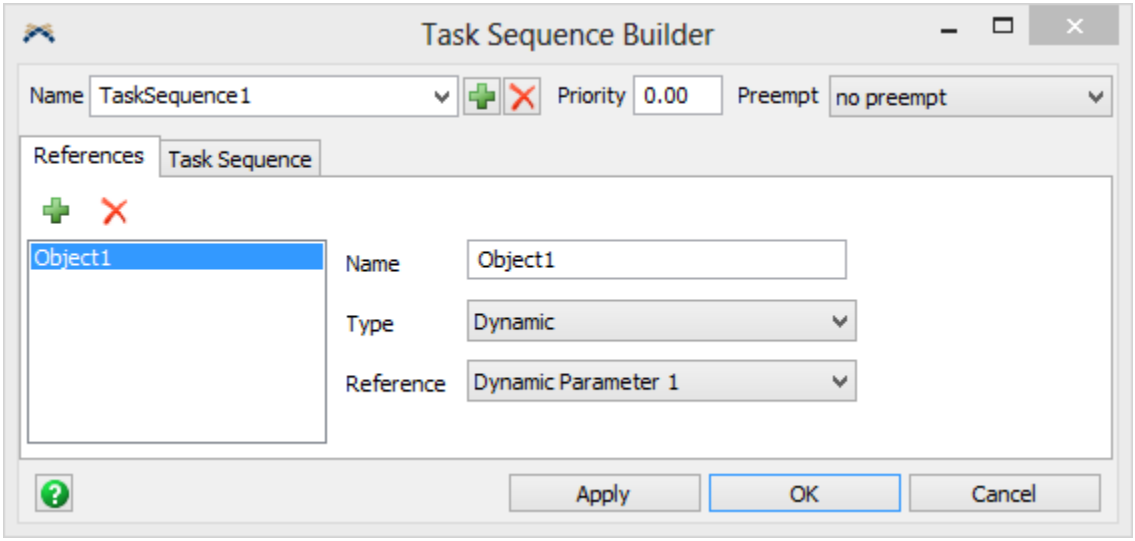
importtable(table, "filename", importcolheaders, importrowheaders) Imports table data from the file specified by filename from a .CSV format.

exporttable(table, "filename") Exports the table in .CSV format to the file specified by filename.

Global Task Sequences

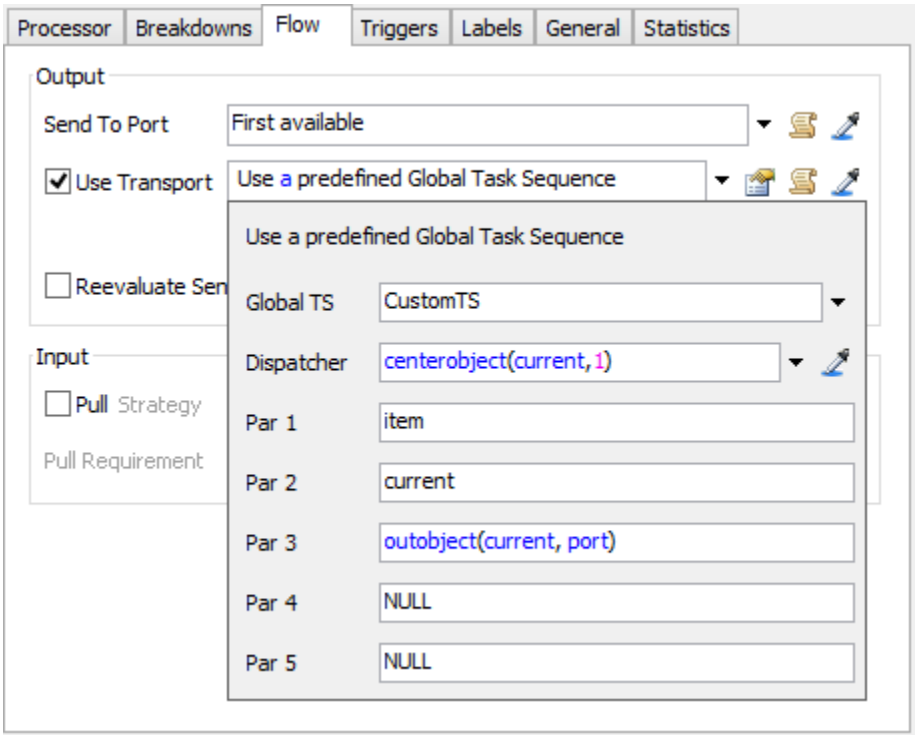
1. Concepts
2. Example

Global Task Sequences Concepts



Global Task Sequences (GTS) are accessed from the Tools menu > **Global Task Sequences**.

Global Task Sequences let you build Task Sequences through a graphical user interface instead of through code. Once you create a GTS, you can dispatch it to a TaskExecuter through the Use Transport picklist of a FixedResource object.



GTS are built by first creating object reference aliases, and then by creating tasks associated with those references. The object references can be either dynamic or static.

- **Dynamic -** Dynamic references are resolved at the time that the instance of the GTS is actually created by being passed in as dynamic parameters to the createglobaltasksequence() command (Par1 - 5 in the image above).
- **Static -** Static references stay the same across all created instances of the GTS.

As seen in the above image, up to five dynamic parameters can be passed into the GTS. These parameters are references to nodes or objects in the model. They allow you to define one GTS that can be used across multiple objects if necessary. A GTS may also have any number of static variables.

Name - The name of the GTS. The combobox has a list of all GTS in the model, allowing you to quickly jump to different GTS.

- Create a new GTS.

- Delete the current GTS.

Priority - This value sets the priority of the GTS Transporters and dispatchers generally sort task sequences so that sequences with higher priorities will be performed first. Task sequences with the same priority will be performed in the order that they were received.

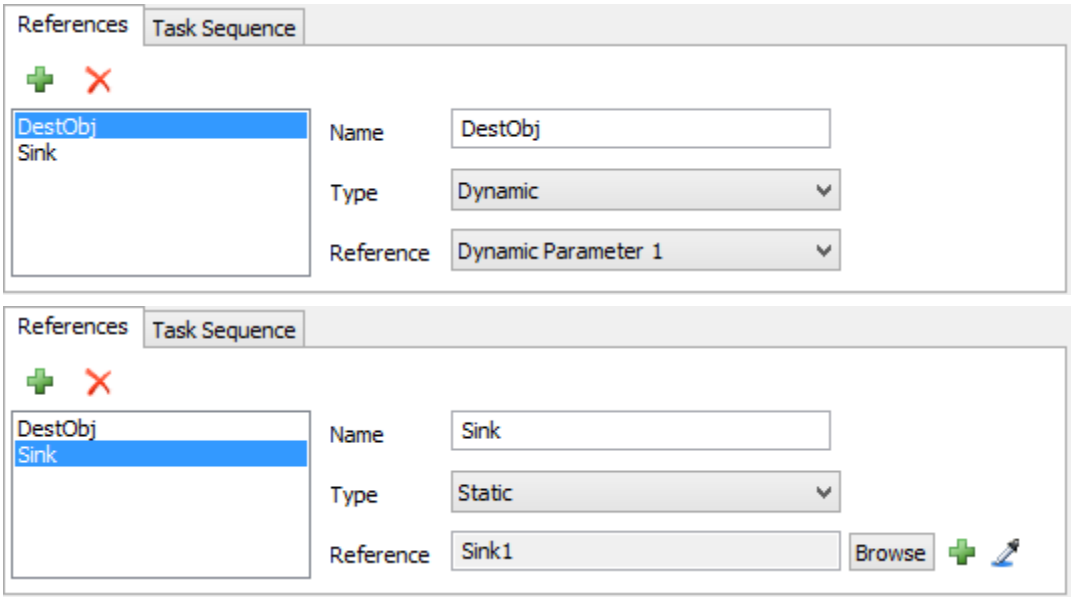
Preempt - If set to one of the preempting values, the task sequences sent to the transporter will automatically preempt whatever the transporter is doing at the time. This may cause the transporter to perform tasks that would normally not be allowed, such as carrying more flowitems than its capacity. For more information on preempting task sequences, see Task Sequence Preempting.

Apply - Saves all changes to the GTS.

OK - Saves all changes to the Time Table and closes the window.

Cancel - Cancels any unsaved changes made to the GTS and closes the window. Please note, there are many parts of the GTS window that apply immediately and are unaffected by pressing Cancel. Adding Tasks and renaming reference variables for example.

References Page



- Adds a new reference.

- Removes the selected reference.

References List - Displays all of the GTS references which can be any number of static parameters, and up to five dynamic parameters.

Name - This is the name of the Reference. This name will be used on the Task Sequence page for defining parameters for each Task Sequence type.

Type - Select this reference to be either Static or Dynamic.

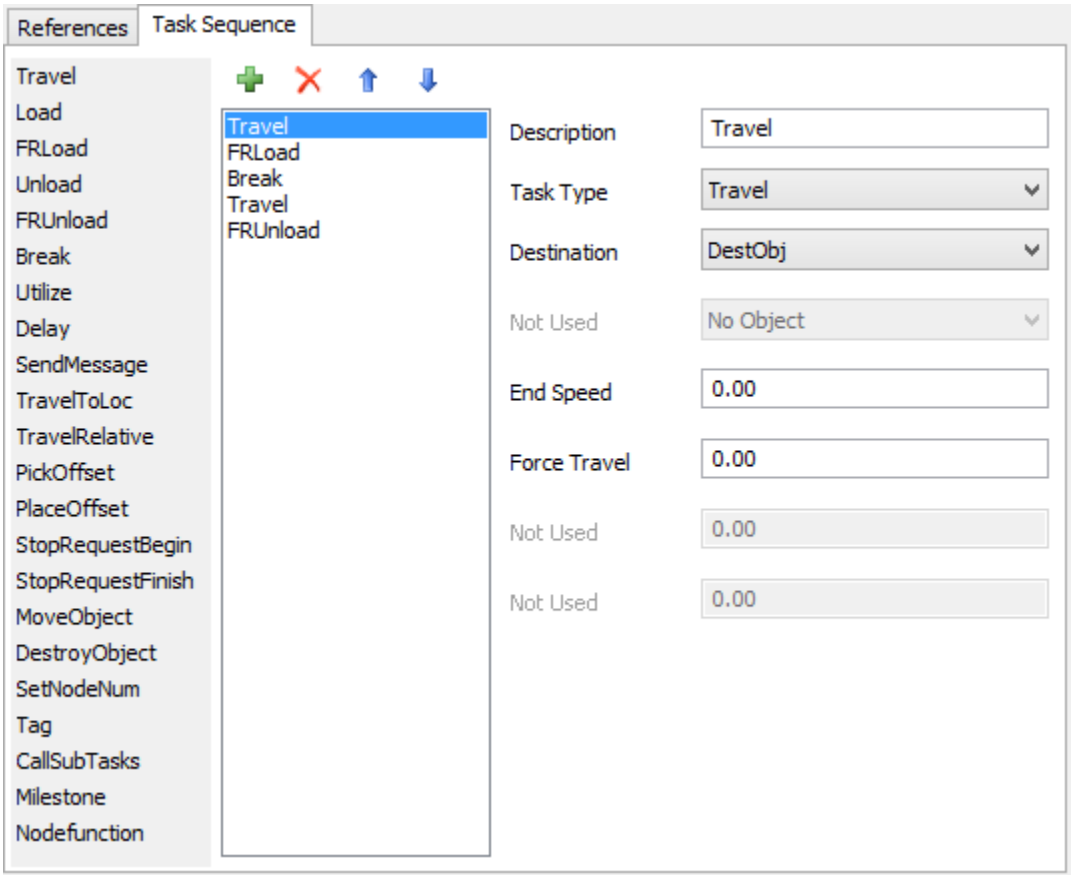
Reference - For Dynamic types, this is a set parameter value from 1-5. For static types, this is the path an object in the model.

- Static type only. Opens a Tree Browse Dialog allowing you to select a node from the tree. Any node or object attribute may be selected.

- Static type only. Opens a popup allowing you to select an object in the model.

- Static type only. Click to enter "Sample" mode, then sample an object, node or attribute in your model.

Task Sequence Page



Tasks List - This is a list of all avliable Task Sequence Types. Drag and drop from this list to the Current Task List to add that task. For more information on these types, see the Task Types page.

- Adds a new Task.

- Removes the selected Task.

- Reorder's Tasks Up or Down in the list.

Current Task List - Displays the set of Tasks for this GTS. Once dispatched, the TaskExecuter will execute each task in order as displayed in this list.

Name - A descriptive name for the Task that will appear in the Current Task List.

Task Type - The Task Type of the currently selected Task.

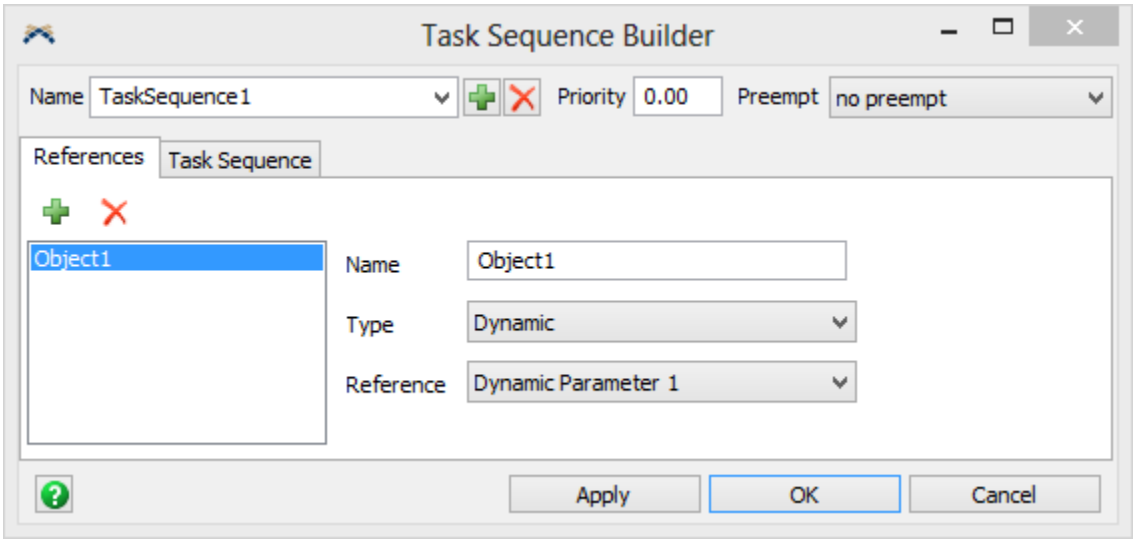
The remaining fields are context sensitive based upon which Task Type is currently selected. The comboboxes like Destination as shown above, contain a list of all Dynamic and Static parameters. The remaining four fields must contain numeric data. They do not execute flexscript!

For more information on how Task Sequences work, refer to the Task Sequences section.

Global Task Sequences Example

Example

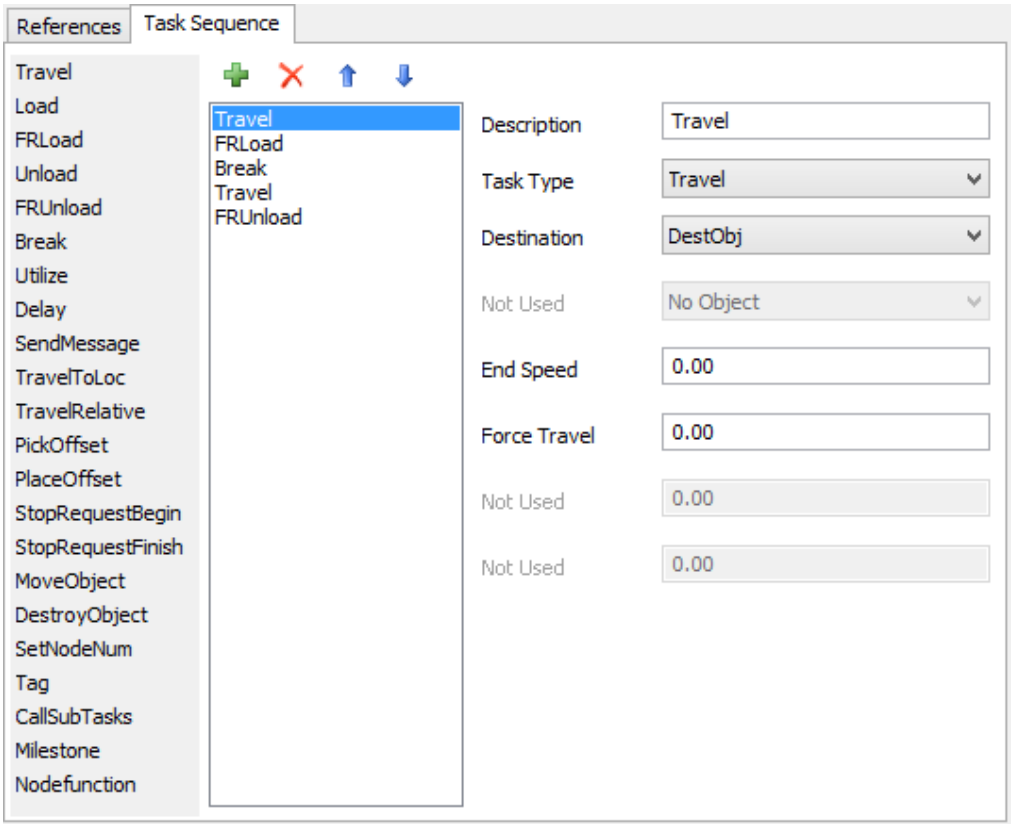
Create a Global Task Sequence (GTS) by selecting the main menu option **Tools** > **Global Task Sequences** > **Add**. The following window will appear:



First, give the GTS an appropriate name.

Press the Add and Remove buttons to add or remove object references. For each object reference, you can give it a name and type (either Dynamic or Static). For dynamic types, you can choose the reference to be one of Dynamic Parameters 1-5. These parameters will be passed in to the createglobaltasksequence() function when the instance is created. For static references, click one of the buttons to the right and select the desired object in the model.

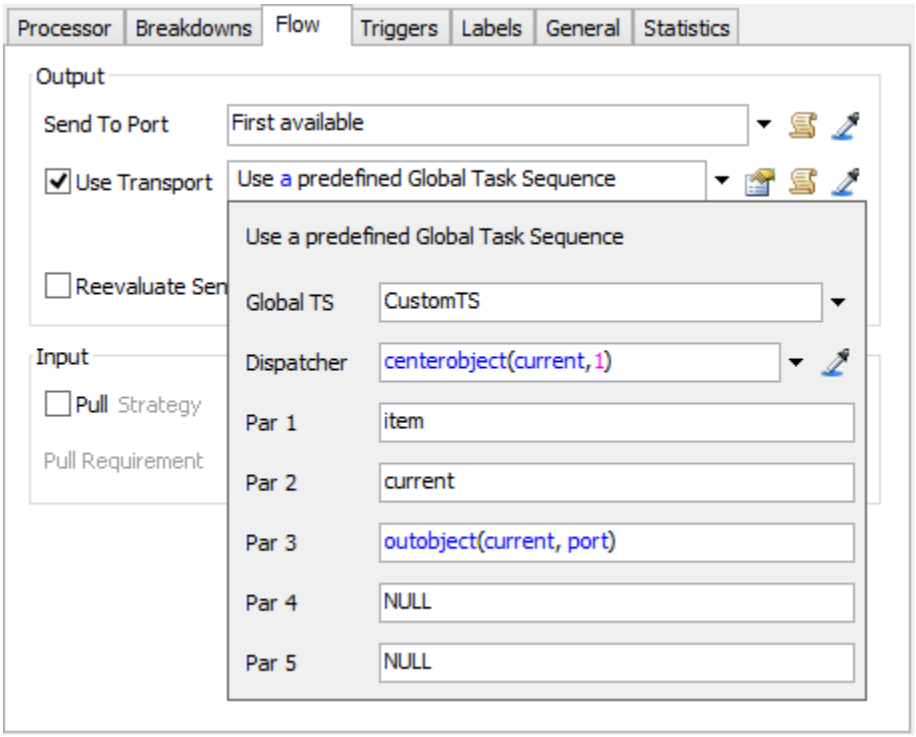
Once you've created the references you need, go to the Task Sequence tab. By default the task sequence is empty. On the left is a grid of droppable tasks. Add tasks by dragging them from the grid on the left and dropping them into the list. An attributes panel will appear to the right of the list.



To edit each task's attributes, just click on the task in the list, and edit the attributes on the right. Use the buttons above the list to reorder or remove the tasks in the sequence.

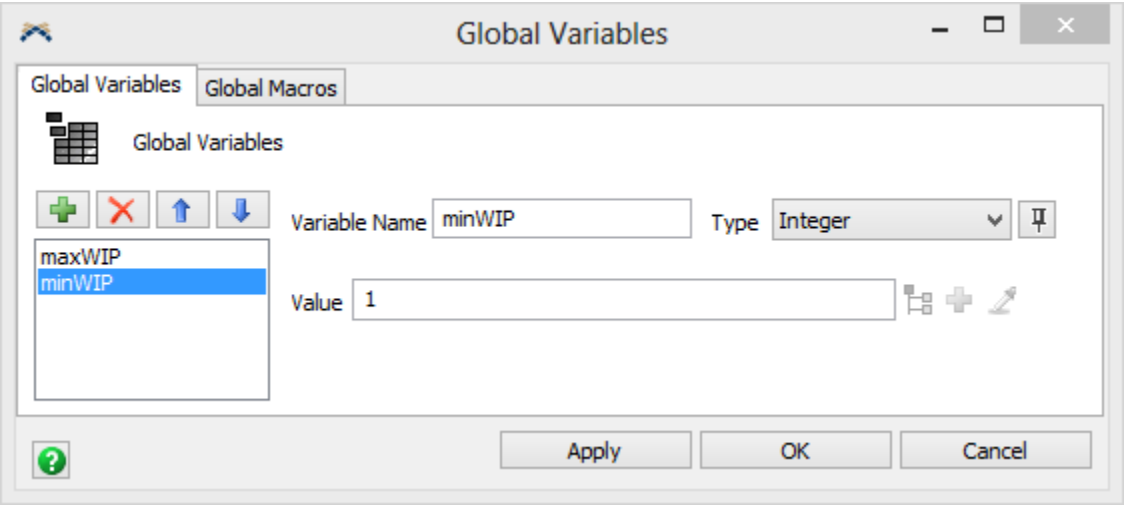
Once you've created your task sequence, you can select the "Create Global Task Sequence" option in the "Request Transport From" field of the Flow tab in any FixedResource Properties window or in the Quick Properties Flow panel.

Just enter the GTS name and the appropriate dynamic parameters.



For more information on how Task Sequences work, refer to the Task Sequences section.

Global Variables



Global Variables can be accessed from the Tools menu > **Global Variables**.

The Global Variables window lets you create global variables and macro definitions that are accessible in FlexScript and C++. Once a variable has been created, you can get and set the value of that variable in a Code Editor window or Script Console.

Note: The value shown is the **initial** value of the variable. It is not the current value of the variable. The current value of the variable is stored in memory and can be seen by returning it in a script window or printing it from somewhere in code. The current value is not stored in the model tree anywhere. Global variable values are reset when you open the model, reset or compile.

There are 8 variable types you can use: integer, double, treenode, string, integer array, double array, treenode array, and string array. For the array types, you can specify the size of the array and the initial value of each array element.

Note on using C++: If you access global variables in C++, you must make sure that the variables' names are globally unique names, meaning you do not use those names anywhere else in your C++ code except for when you are accessing the global variables themselves. FlexSim uses a macro definition to define these variables, so any other occurrences of the variable name may cause model malfunction and compile errors.

- Adds a new Global Variable.

- Removes the selected Global Variable.

- Reorder's the selected Global Variables Up or Down in the list.

Variable List - Displays all the model's Global Variables. Click to edit.

Variable Name - The name of the Global Variable. This is the name that will be used when writing code, ie `setlabelnum(current, "wIP", maxWIP).`

Type - Specify the Global Variable's type.

- Only available for Integer and Double types. Pins the global variable to a Dashboard as either the current value, bar chart or line graph.

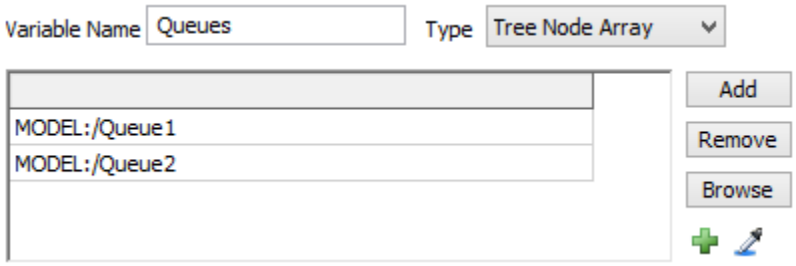
Value - The initial value of the Global Variable.

- Only available for Tree Node type. Opens a Tree Browse Dialog allowing you to select a node from the tree. Any node or object attribute may be selected.

- Only available for Tree Node type. Opens a popup allowing you to select an object in the model.

- Only available for Tree Node type. Click to enter "Sample" mode, then sample an object, node or attribute in your model.

If an array type is chosen, the Global Variables window will display the following:



Add - Adds an empty value to the end of the array.

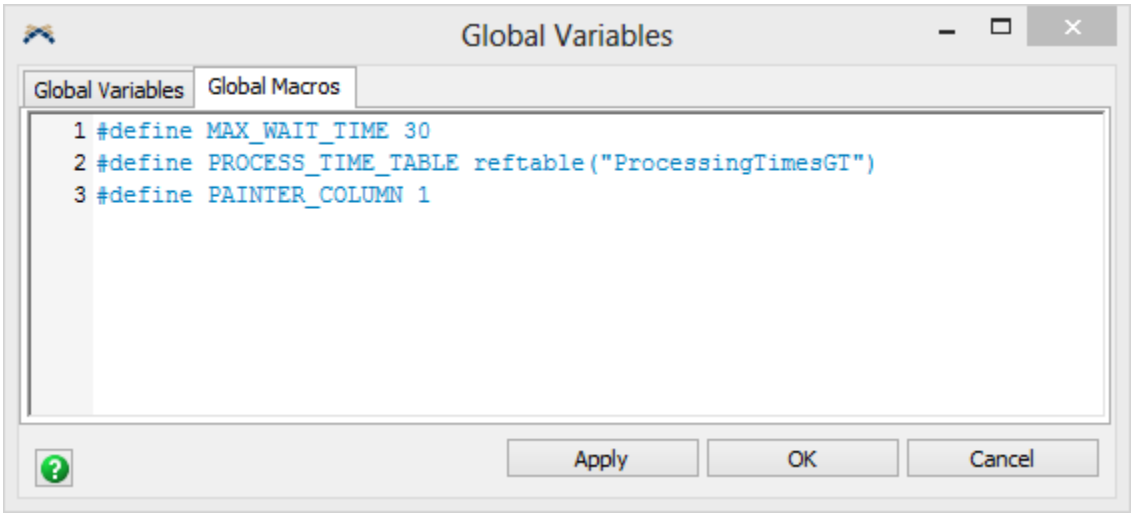
Remove - Removes the selected value.

Browse - Only available for Tree Node type. Opens a Tree Browse Dialog allowing you to select a node from the tree. Any node or object attribute may be selected.

- Only available for Tree Node type. Opens a popup allowing you to select an object(s) in the model. If a value is selected in the list, this sets that value, otherwise, it adds new values to the end of the array.

- Only available for Tree Node type. Click to enter "Sample" mode, then sample an object, node or attribute in your model. If a value is selected in the list, this sets that value, otherwise, it adds a new value to the end of the array.

Global Macros



The global macros page lets you make macro definitions.

You can define macros using #define statements, as follows:

```
#define MAX_WAIT_TIME 30
#define PROCESS_TIME_TABLE reftable("ProcessingTimesGT");
#define PAINTER_COLUMN 1
```

Once you have made these definitions, you can use them in your code:

```
gettablenum(PROCESS_TIME_TABLE, 1, PAINTER_COLUMN)
```

Note: Macro definitions do not end with a semicolon. If you put a semicolon in the macro definition, it may do things you don't expect it to do. Macros essentially replace the given text with the following specified value/text throughout your code. If you have a semicolon at the end of the statement, you may end up with semicolons in incorrect places in your code.

Graphical User Interfaces

1. Concepts
2. Example
3. Reference
    - GUI Events and View Attributes
    - View Attributes Reference

Graphical User Interfaces Concepts

Topics

- **Why Create GUIs?**
- **GUI Views**
- **Designing a GUI**
- **GUI Building Tips**
- **Working with the GUI Editor**
- **Traversing the Tree**
- **Attribute Lists**
- **Tree View - Viewing Attributes vs. View Structure**
- **Linking to the Model**
- **Copy an Existing GUI**

Graphical User Interfaces (GUIs) are accessed from the Tools menu > **Graphical User Interfaces**.

GUIs allow you to create your own window interfaces for your model and its objects. A GUI can communicate with any object in the model, in any way you want.

GUIs are so common that there is a button in the toolbar to open such a window.

In FlexSim, GUIs are stored as a node with sub-nodes in the tree. Each node represents a part of your GUI. The attributes in the object data of each of those nodes represent variables that affect how that part of your GUI works. When a GUI is opened, it creates a new node in VIEW:/active that is a copy.

Why Create GUIs?

- Not everyone knows how to navigate FlexSim. You can give others the ability to manipulate certain parts of your model without having to know FlexSim.
- Save time on extensive testing. GUIs can help you to change parameters quickly in your model during the testing process.
- GUIs look professional.

GUI Views

FlexSim GUIs are made up of building blocks called views. Views are windows that perform specialized roles and can be combined hierarchically. These views will allow you to view and manipulate data in the FlexSim tree structure. Since data takes many forms, there are many types of views.
Below is a list of the View Types available in FlexSim. The number next to each view type is the viewwindowtype. This number is a direct reference to the type of view and will not change in future releases of FlexSim.

For more information on GUI views, see the View Attributes Reference page.

View Types			
Windows Common Controls		**FlexSim Registered Controls**	
Static (or label) Button	103	Dialog Panel	4 102
Radiobutton	100	Groupbox	107
Checkbox	106	Table	5
Edit	105	Graph	6
Trackbar	101	3D View	2

Combobox	122	Tree	0
Listbox	109	IconGrid	7
Tabcontrol	114	Script	8
Scrollbar	115	HTML	124
Statusbar	104		
Spinner	110		
DateTimePicker	123		
Treeview	125		
	119		

Creating a Modal Window: To create a modal window, use viewwindowtype 4 (Dialog) and add the subnode *FS_MODAL* into the object's style attribute.

Designing a GUI

There are whole books written about the philosphy of GUI design. The major topic, ease of use. Avoid the temptation of making a GUI window try to do too much. GUIs should be simple to navigate and focused in their function.

It is recommended to sketch out what you want a GUI window to look like before you open the GUI editor.

GUI Building Tips

It is important to have a good Control naming convention. A good short name will make referencing the object in code much easier.

Use Panel controls (invisible and not invisible) to group, move, and copy sections of your GUI.

GUI building essentially consists of adding views to your GUI window and giving those views the appropriate attributes. While the "View Specific" option of the attribute list gives you some good hints of which attributes are appropriate to add, there are also other ways you can become more comfortable and experienced building GUIs.

Another way you can become more familiar with building GUIs is by simple experimentation. Add an attribute that you've never added or seen before and see how it affects the GUI. If there is no change, then either the attribute doesn't do anything for this type of view, or it's somehow not implemented right. Fiddle around with different settings and values for the attribute to see if and how it changes the GUI. If still nothing changes, then just move on and try something else. If it does, great! That's one more tool that you have in your knowledge base.

Practice, practice, practice. As you continue to build more and more GUIs, the speed and efficiency with which you do it will increase significantly, and you will be able to get a feel for what GUI styles are more user friendly.

Working with the GUI Editor

Here are some tips with working with the GUI editor:

- Think about not only how to arrange the controls in the GUI canvas (window editor) but also in the tree. Having familiarity with the Tree is a requirement for good GUI building.
- Moving controls in the tree can be done using the Edit Highlighted Object and Tree Movement fields in the GUI builder window.

- The attribute nodes of the GUI controls define how the control will look/behave. Each control has a default set of attribute nodes but others can be added from the Controls palette on the GUI builder window.

Traversing the Tree

@ : This symbol tells the traversal to go to the owner view of the current node. For this coldlink node, it is the main GUI window.

> : This symbol tells the traversal to go into a node's attribute tree.

+ : This symbol tells the traversal to read the current node's text as a path to an object, and go to that object.

***/* :** This symbol tells the traversal to go into the current node's sub-tree.

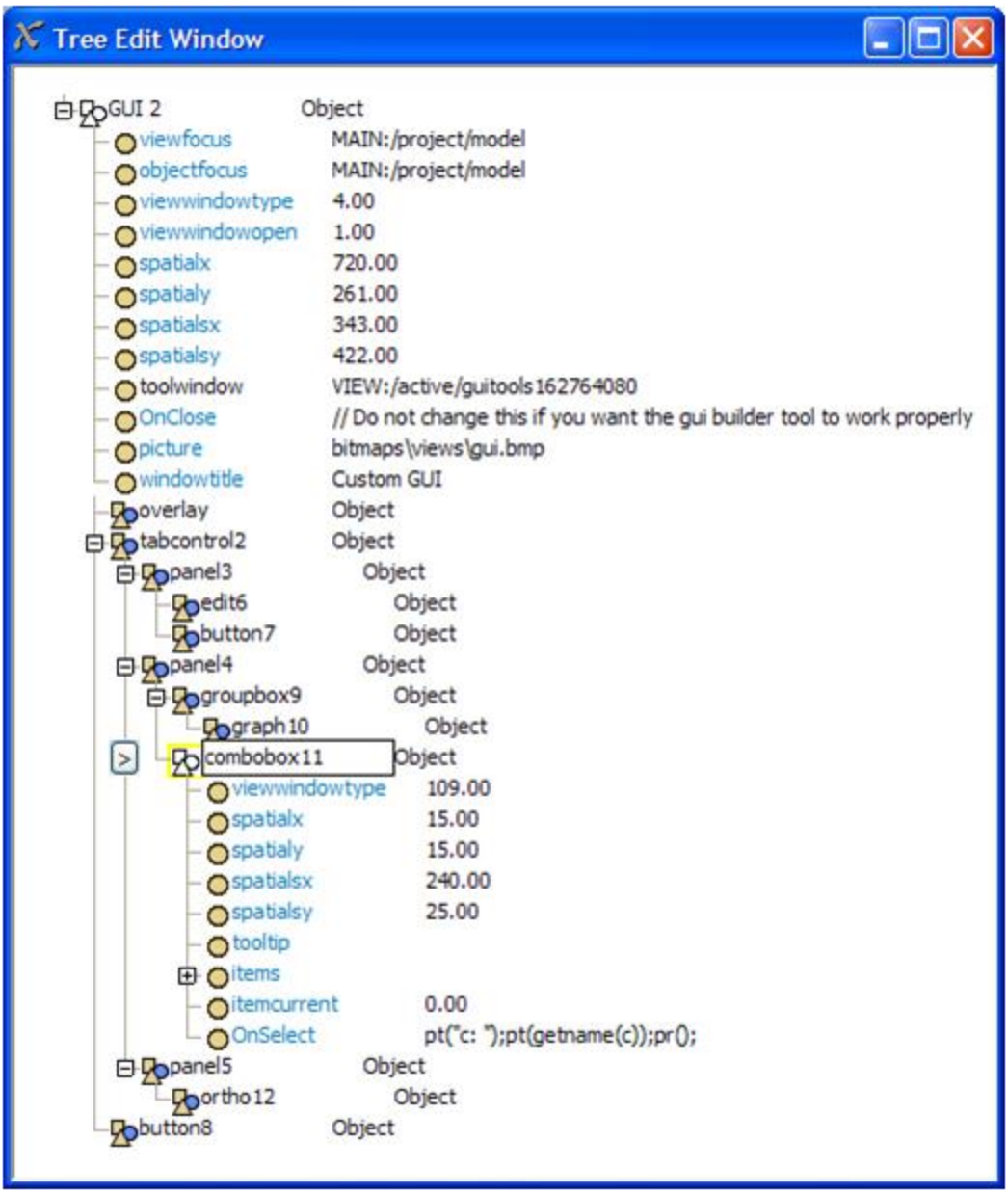
.. : This symbol tells the traversal to go to the current node's parent tree, or up one level.

? : This symbol causes a recursive tree search for the subsequent name. For example, node("/?FlowItemBin", model()) will search for a node with the name FlowItemBin in the model tree. This returns the same node reference as an explicit path definition would: node("/Tools/FlowItemBin", model()).

$flexscript$: This syntax allows you to enter in custom flexscript code when accessing the path. For example, if you had a Global Macro `#define MY_NODE_RANK 4` then having a path of *>objectfocus+/MY_NODE_RANK* would be read as *>objectfocus+/4*. You can execute any valid flexscript code. Another example, if you have an object in the model with a label called focus that contains "Tool4", then the following path *>objectfocus+/$getlabelstr(node("MyObject", model()), "focus")$* would be read as *>objectfocus+/Tool4*.

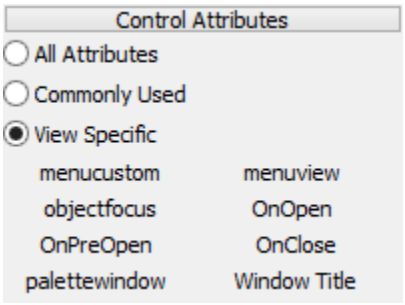
Below is a example table for Tree Referencing (Tree shown below table)

Code:	Returns:
c	combobox11
itemcurrent(c)	itemcurrent
node(">itemcurrent",c)	itemcurrent
ownerobject(c)	panel4
node("..",c)	panel4
node("../../panel5/ortho12",c)	ortho12
ownerview(c)	GUI 2
node("@",c)	GUI 2
node("@>toolwindow",c)	toolwindow
node("@/tabcontrol2/panel3/1",c)	edit6
node("../../3/1",c)	ortho12
node("@/button8",c)	button8
node("@>objectfocus+",c)	model



Attribute Lists

The list of possible attributes to the left of the GUI builder's tree view has three options for viewing attribute lists. They are: all attributes, commonly used attributes and view specific attributes.



All Attributes - If this option is selected, all possible attributes are displayed. Usually you will not need to use this option.

Commonly Used - If this option is selected, a list of commonly used attributes will be shown. These include things like alignment attributes, which allow you to anchor a view's position or size to the right or bottom of its container view.

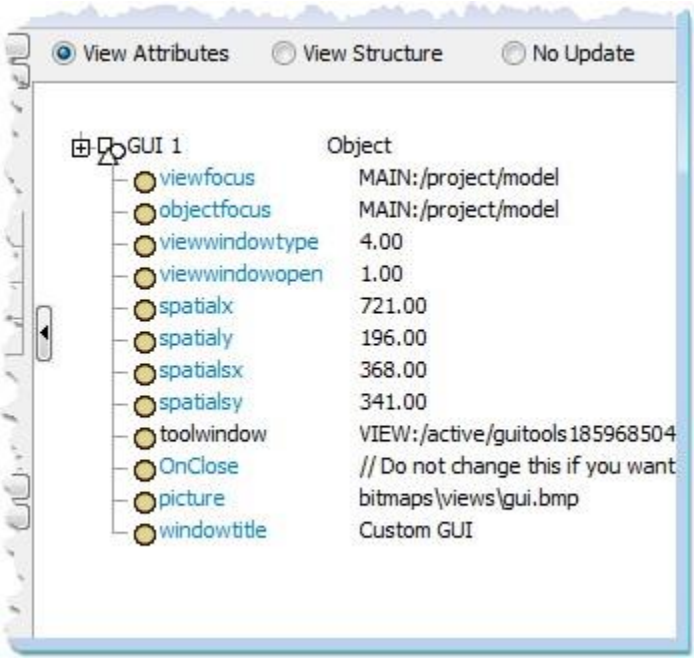
View Specific - This option is the default option. If it is selected, then the attribute list that is shown will be specific to the type of view, or view, that you are currently editing. If you click on a button view, an attribute list will appear that is specific to a button. This includes an OnPress attribute, which is executed when the button is pressed. A label view will have a different set of attributes that are used for it, etc.

To see a detailed list of attributes and their usage by control type, go to View Attributes Reference.

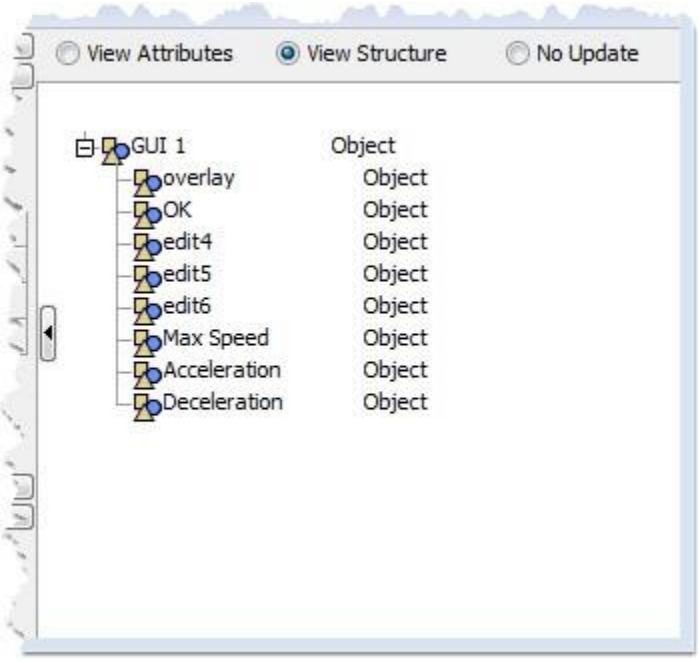
Tree View - Viewing Attributes vs. View Structure

The GUI builder's tree view also has two options for viewing the tree. They are: view attributes and view structure.

View Attributes - This option will view the attributes of the currently selected view.



View Structure - This option will view the tree structure of the currently selected view. This is useful for rearranging and editing the structure of the GUI's tree.



No Update - This option will cause the GUI builder to not update the tree focus when you click on an control in the gui canvas. Some users prefer this as it doesn't change the view every time you click in the canvas.

Linking to the Model

FlexSim uses two types of linking when tieing a GUI object into the model. Coldlinks, and hotlinks.

coldlinks

A coldlink attribute tells the view's text field to be linked to a certain node in the object's attribute tree. The link is "cold" because it gets the value only once when the window opens, and sets the value only when an apply button is pressed.
Example: On the Processor tab of the Processor, the Max Content field is an example of a coldink.

hotlinks

A "hot" link, on the other hand, would continuously update its text field as the value in the model changes.
Example: On the statistics page of the Processor, the Content statistics are examples of a hotlink.

hotlinkx/coldlinkx

Similar to the above, but rather than using a special string of characters, FlexScript code is allowed to establish the link. The "c" passed in as argument 2 of the below example represents the owner object of the attribute.
Example: return(node("@>objectfocus+",c));

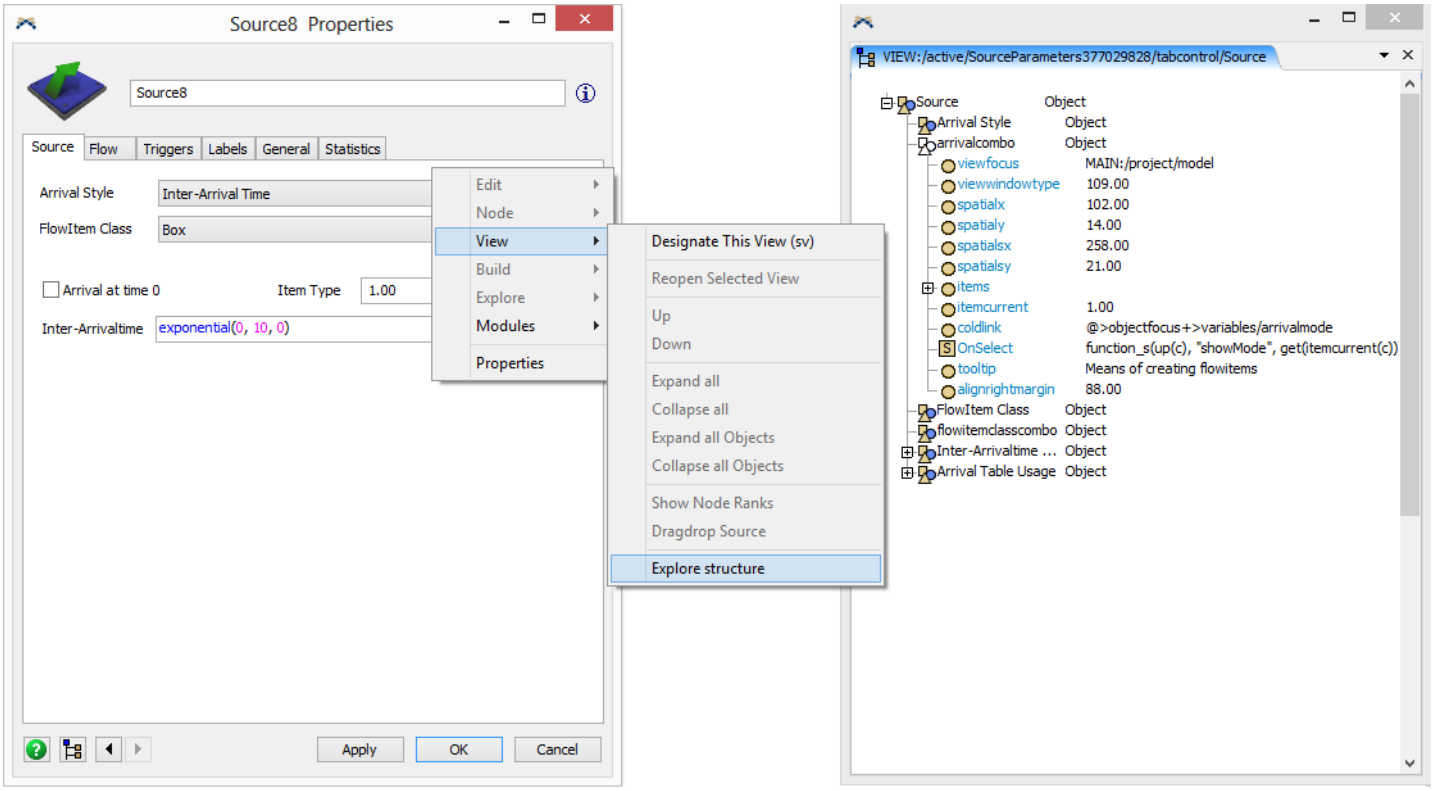
Copy an Existing GUI

If you see a window view or GUI that you want to copy verbatim from an existing window to your GUI there are two ways to copy.

Copy from Tree
Find that view in the tree, right click on it and select "Edit > Copy"

Copy from Window
Open up the window you want to copy. Right click on it, and select "View > Explore Structure" from the popup menu. This will open a tree window that shows the structure of the window. From here you can view the attributes of that window to see which attributes are present, so that you can add those attributes to your own GUI. Right click on the view node and select "Edit > Copy"

Now go to your own GUI's tree structure. In the GUI's tree structure, create a new blank node by right clicking on the container view you want to place it in, and select "Node > Insert Into". Then right-click on the new blank node, and select "Edit > Paste". This will paste the view into your GUI. Press F5 to refresh the view with its added view.


Source8 Properties
Source8
Source
Flow
Triggers
Labels
General
Statistics
Arrival Style
Inter-Arrival Time
FlowItem Class
Box
Arrival at time 0
Item Type
1.00
Inter-Arrivaltime
exponential(0, 10, 0)
Edit
Node
View
Build
Explore
Modules
Properties
Designate This View (sv)
Reopen Selected View
Up
Down
Expand all
Collapse all
Expand all Objects
Collapse all Objects
Show Node Ranks
Dragdrop Source
Explore structure
Apply
OK
Cancel
VIEW:/active/SourceParameters377029828/tabcontrol/Source
Source Object
Arrival Style Object
arrivalcombo Object
viewfocus MAIN:/project/model
viewwindowtype 109.00
spatialx 102.00
spatialy 14.00
spatialsx 258.00
spatialsy 21.00
items
itemcurrent 1.00
coldlink @>objectfocus+>variables/arrivalmode
OnSelect function_s(up(c), "showMode", get(itemcurrent(c))
tooltip Means of creating flowitems
alignrightmargin 88.00
FlowItem Class Object
flowitemclasscombo Object
Inter-Arrivaltime ... Object
Arrival Table Usage Object

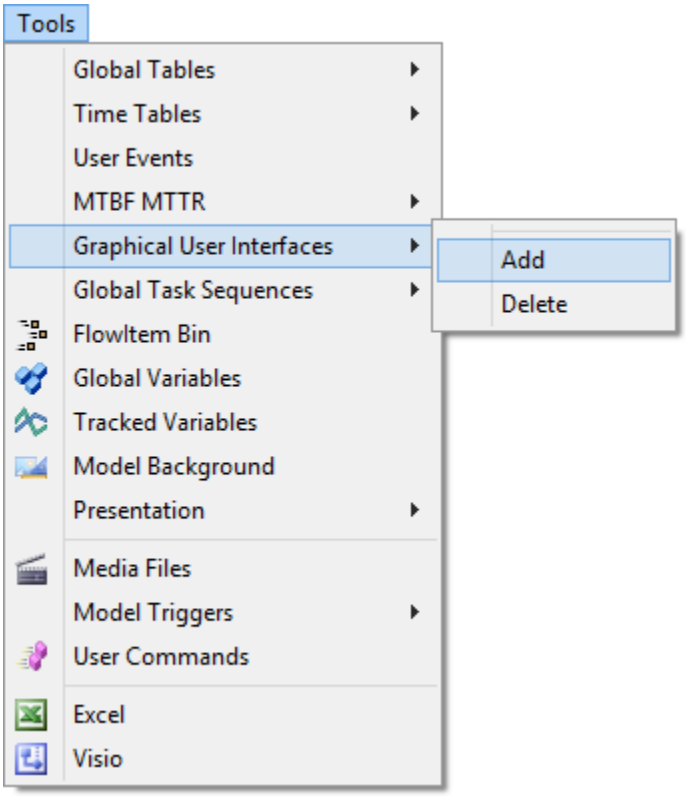
Graphical User Interfaces Example

Topics

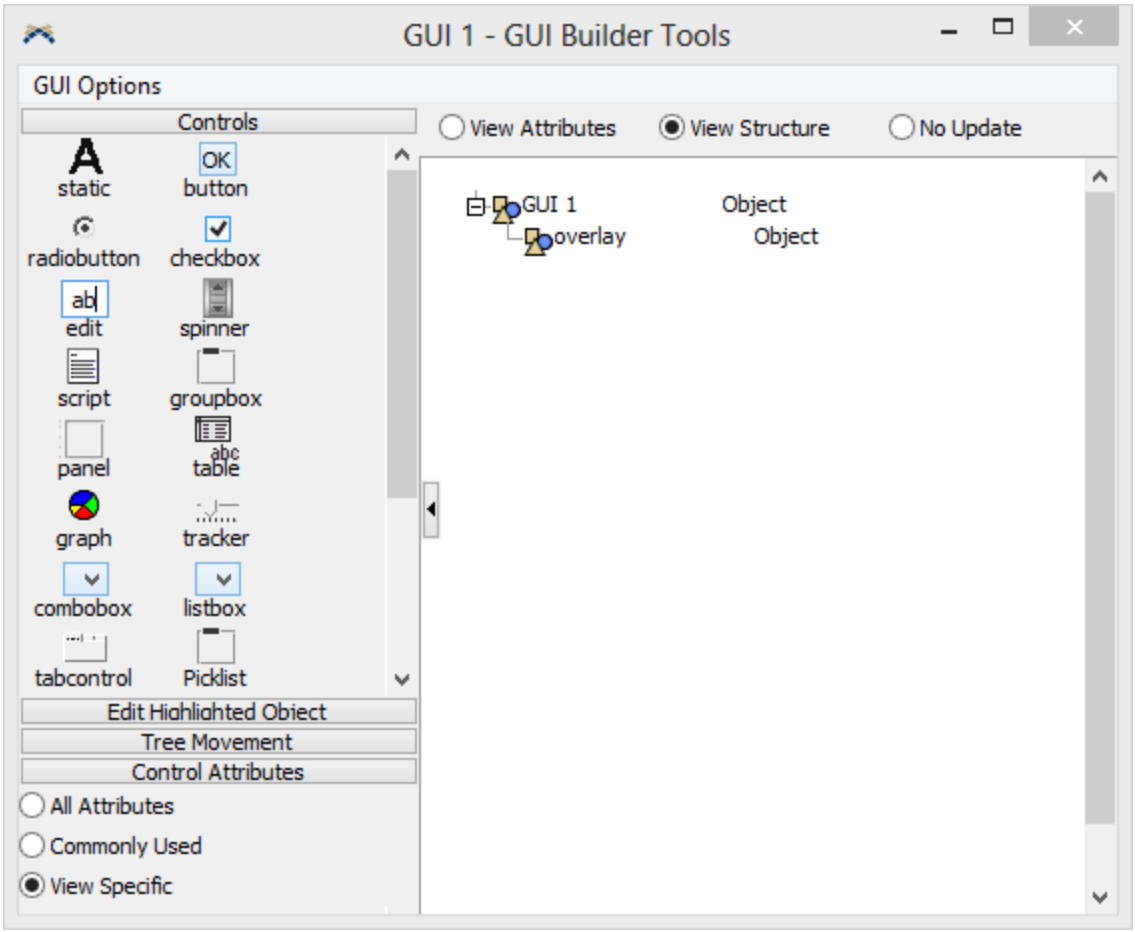
- **Building a GUI**
- **Change View Names**
- **Link the Edit Views**
- **Direct Model Objects to This GUI**

Building a GUI

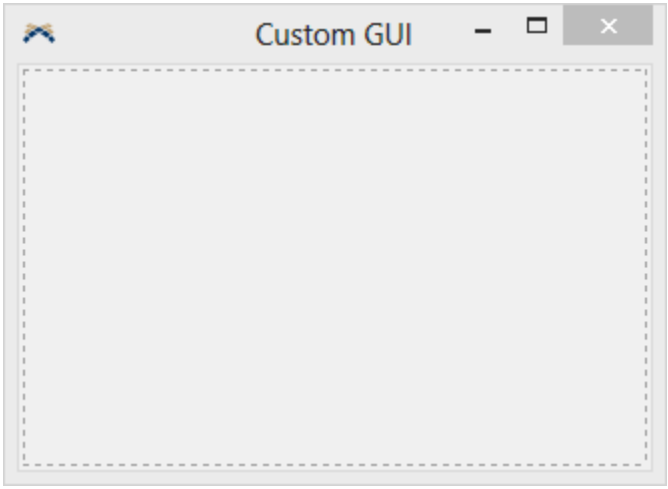
Add a GUI by going to the **Tools Menu > Graphical User Interfaces > Add**.



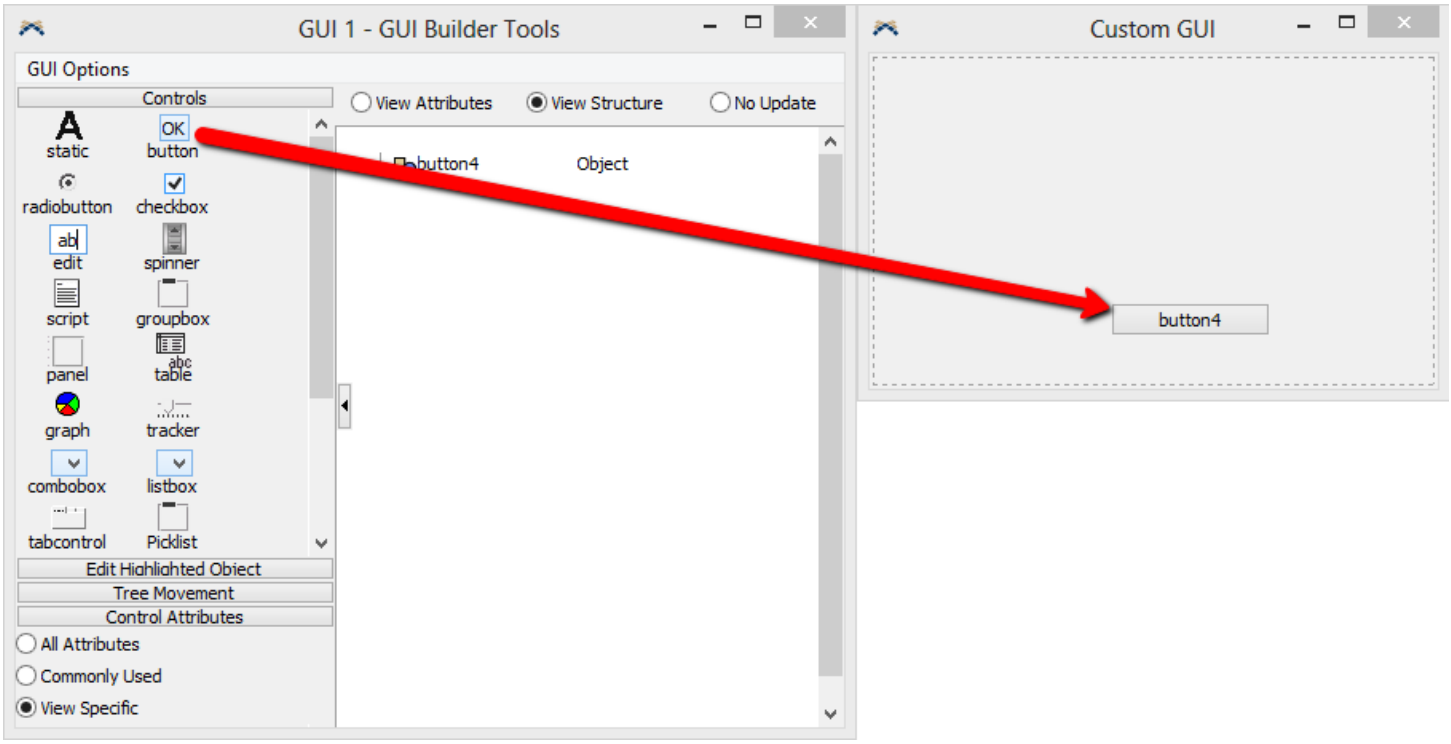
Once you have added a GUI, two windows will open. The window on the left is called the GUI builder. This window provides you with several tools for building your GUI.



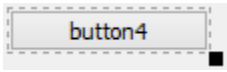
The window on the right is called the GUI canvas. This window shows what your GUI looks like. Initially it is blank. We will add GUI views to it in a drag-drop fashion from the GUI builder window.



As an example, we will create a simple GUI that allows you to edit the max speed, acceleration, and deceleration variables of an Operator object. First, let's drag a few simple views onto our GUI canvas. From the top panel of the GUI builder, drag a button onto the GUI canvas.

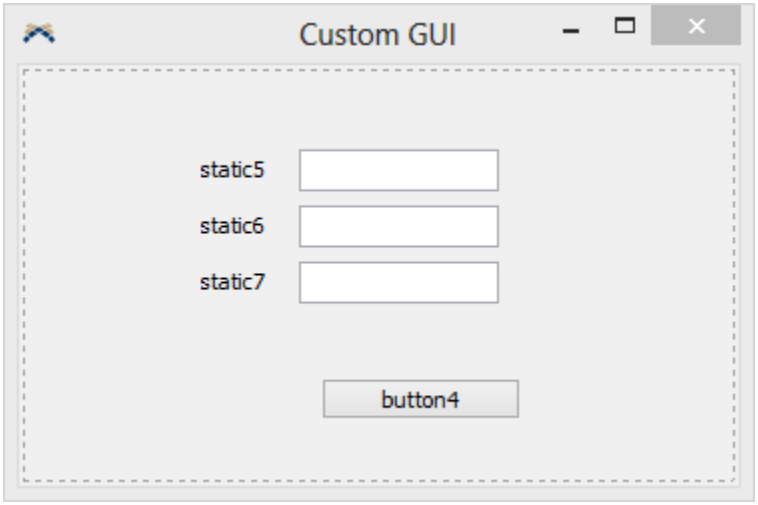


You can now select the button by clicking on it. When the button is selected, you will see a dotted outline around it, as well as a black square to the bottom right of it.



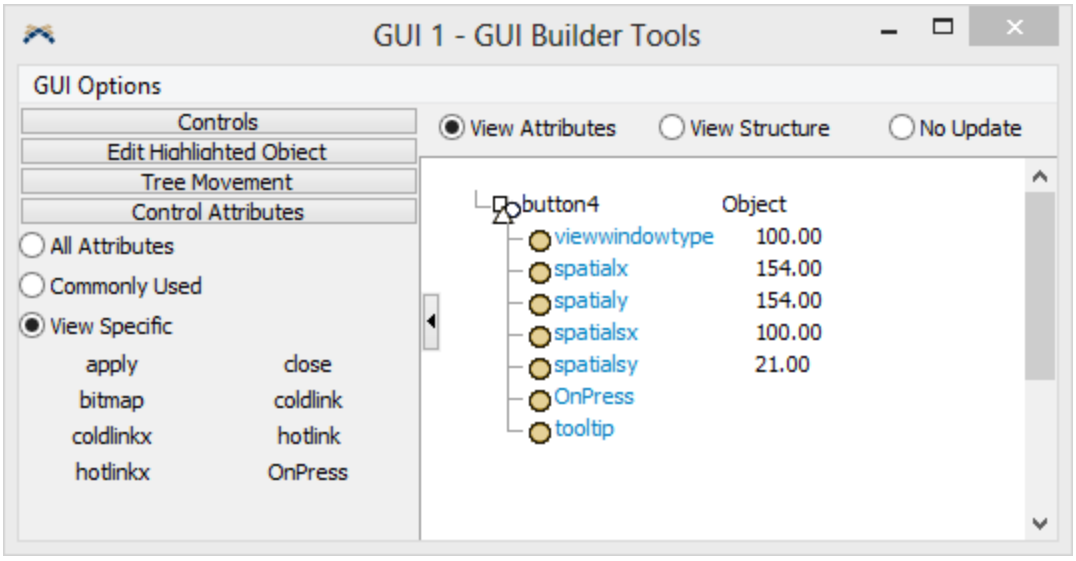
To move the button, just click and drag the button to the location you want it at. To change the size of the button, click and drag the black square.

Now add three static views and three edit views onto the GUI canvas, as shown below.

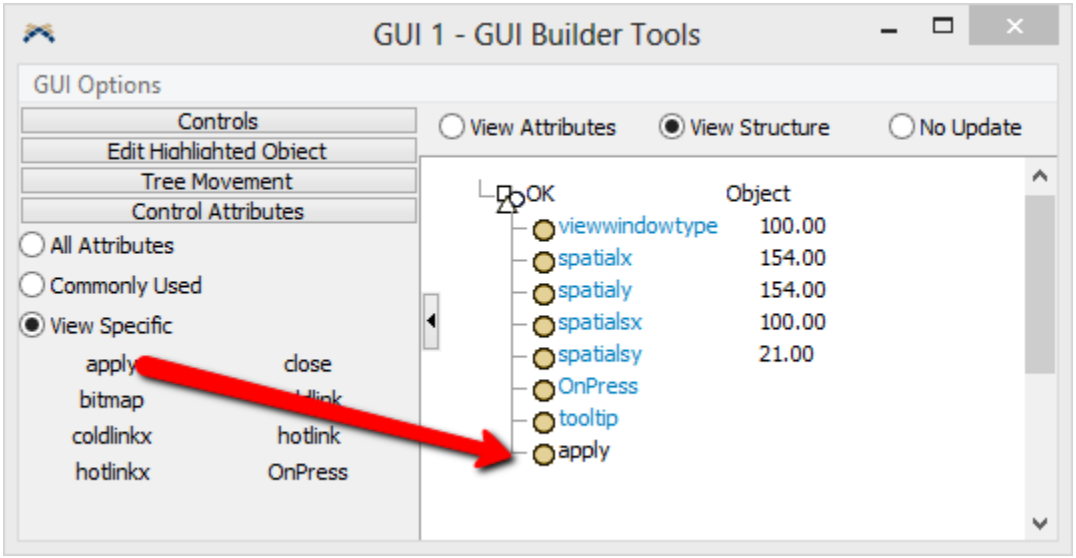


Change View Names

Now we want to change the names and attributes of our GUI views. Collapse the controls toolbar by pressing on the "Controls" button in the gui builder. First, let's make the button an OK button that will apply the edits the user has made, and then close the window. Click on the button. Then click on the "View Attributes" radio button in the gui builder window. Notice that the button and its attributes are now shown in the tree view of the GUI builder.



Click on the button name ("button4" in our case) and rename it to "OK". On the left are some commonly used attributes that can be added to the button. To add an attribute, just drag it from the icon grid on the left to a blank area in the tree view.



Add an apply attribute and a close attribute to the button. The apply attribute causes all coldlinks and hotlinks found in the view to be applied to their respective destination nodes when the button is pressed. Coldlinks and hotlinks will be explained later. The close attribute causes the window to close when this button is pushed.

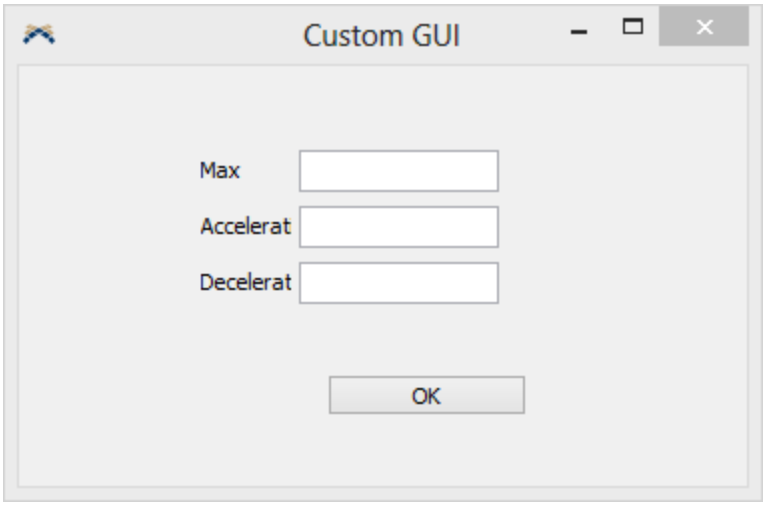
Alternative Method - Alternatively, instead of adding the apply and close attributes, you could add the following code the the OnPress attribute of the button:

```
applylinks(ownerview(c));
postclosewindowmessage(ownerview(c));
```

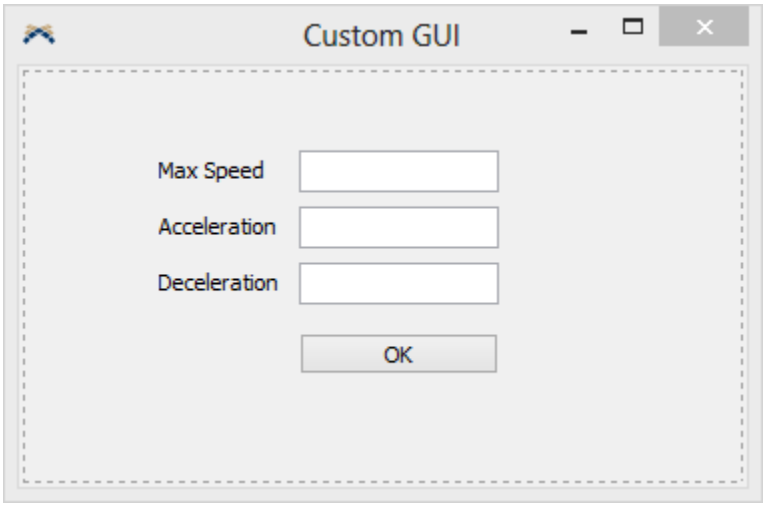
This code would have the same effect as the apply and close attributes.

Now click on the first static view. Again, the static view and its attributes should appear in the tree view of the GUI builder. For the name of the first static view, enter "Max Speed". You will not see the name change on the static view until the view is refreshed. Now click on the second static view and set its name to "Acceleration". Then click on the last static view and change its name to "Deceleration".

Now that you have made a few changes to the GUI views we will refresh the GUI canvas to see these changes. Click on the GUI canvas window and then press the F5 button. This will change the GUI canvas from editing mode to regular viewing mode. Now your window should look like a normal window without the dotted lines around it.

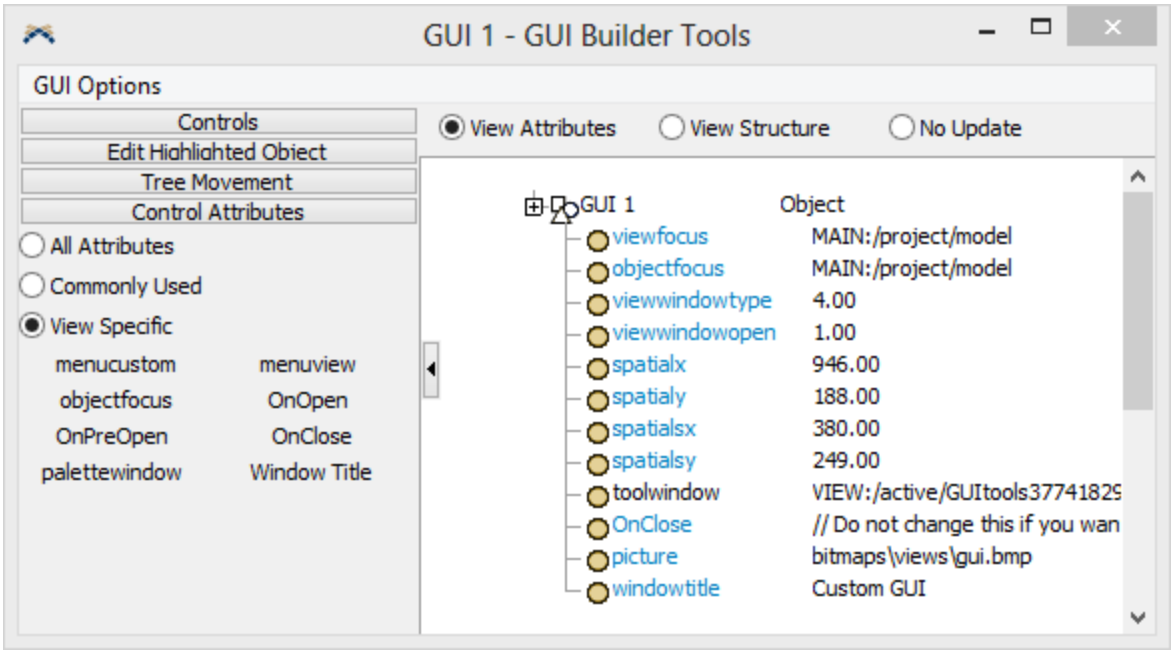


Notice that the label views are now not large enough to fit the text that they are showing. To fix this, go back into editing mode by selecting the GUI canvas window and pressing F5 again. Now rearrange the sizes and locations of your views so that there is enough room to show the entire text of the labels.



Link the Edit Views

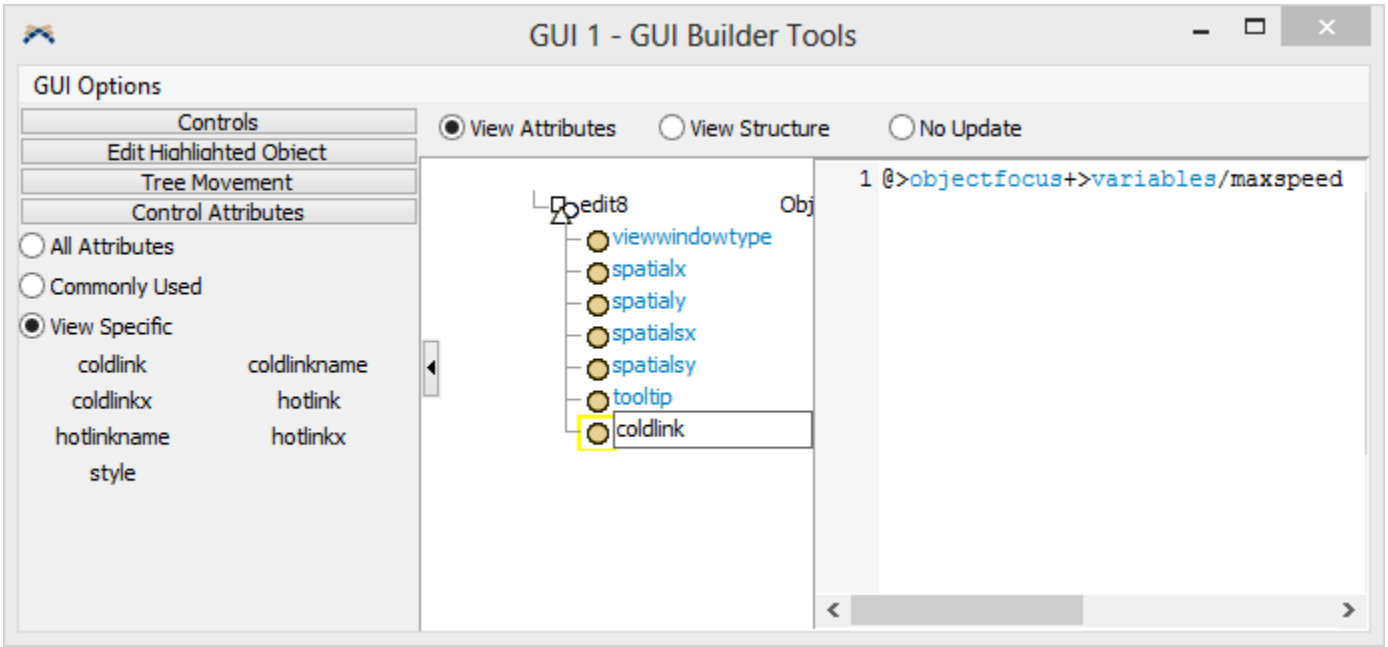
Now let's connect the edit views up to their proper nodes in the model. First let's explain some concepts. Click in a blank area in your GUI canvas. This will cause the tree view of the GUI builder to show the attributes of the main GUI window.



Notice that there is an attribute called objectfocus. Right now this attribute shows the path "MAIN:/project/model". Later on, though, once you've associated an Operator with this GUI, and then double-click on the Operator to open this window, the objectfocus attribute will be changed. It will specify a path to the Operator you are editing. For example, you've associated an Operator named Bob with this GUI. When you double-click on Bob, an instance of this GUI is created, and its objectfocus attribute is set to the string path: "MAIN:/project/model/Bob". This is important to know when creating edit fields that are linked to our object. Now let's go back to the first edit field.

Click on the first edit view to view its attributes in the tree. Add a coldlink attribute from the list on the left.

Now enter the following as the text of the coldlink attribute: @>objectfocus+>variables/maxspeed



Links should be made to a node that contains some desired value. Sometimes linking directly to the object property node is undesireable, such as code nodes. In this case, link to a label and use the label value as part of the code property (use executestring()).

The coldlink/hotlink text specifies a path to a node that the edit field is to be associated with. This path starts at the coldlink node itself, and should specify a path to the maxspeed variable node on the operator. The different symbols in the path are ways of specifying how to traverse the tree to the destination node.

The coldlink you have specified does the following:

1. Starting at the coldlink node, go to its ownerview, or the main GUI window (@).

2. From there, go into its attribute tree and find the attribute named objectfocus (>objectfocus).

3. Interpret the text of the objectfocus node as a path to a node, and go to that node (+). Remember that when we open this window for our Bob operator example, the objectfocus attribute will be changed to "MAIN:/project/model/Bob". So it will now go to our Bob operator in the model.

4. From there, go into the object's (Bob's) attribute tree, and find the node name variables (>variables).

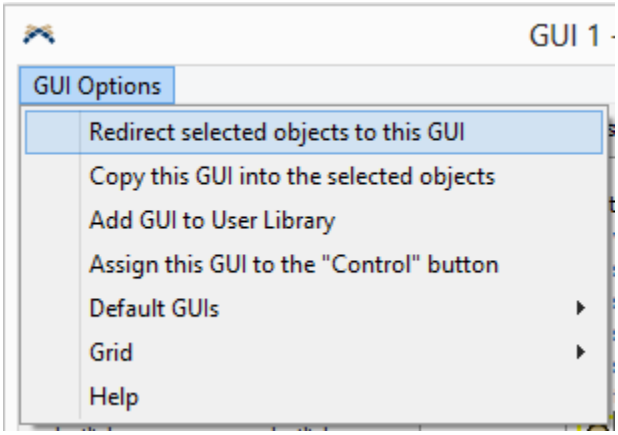
5. From there, go into the node's sub-tree and find the node named maxspeed (/maxspeed).

Now connect the other two edit views. Click on the second edit view, then in the tree view add a coldlink attribute and specify its text as: @>objectfocus+>variables/acceleration. Do the same for the last edit view and set its coldlink attribute to: @>objectfocus+>variables/deceleration.

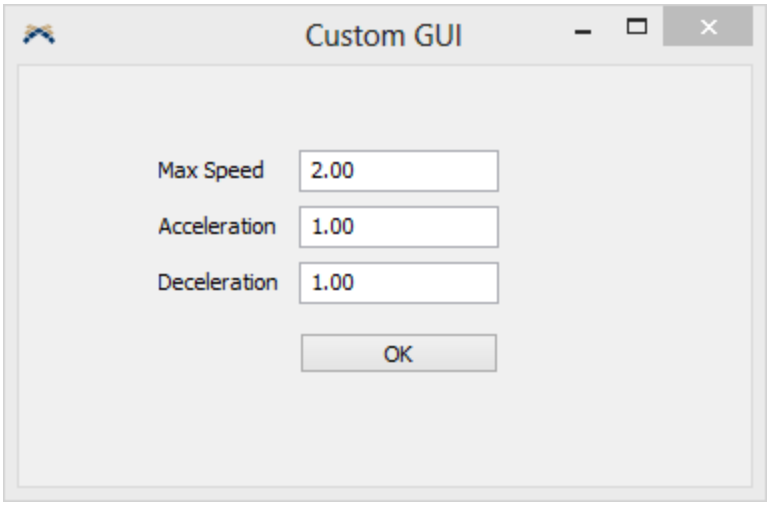
Direct Model Objects to This GUI

Now that we've finished creating our GUI, let's direct an operator in our model to this GUI. Drag an operator into your model. Select it by holding Control or Shift down and clicking on the operator.

Now go back to the GUI builder window, and from its menu, select "GUI Options > Redirect Selected Objects to this GUI".



This will redirect all selected objects in the model to use this GUI instead of their usual properties window. Now close the GUI builder window. This will automatically close the GUI canvas window as well. Now double click on the operator you have selected. This will open the GUI window you have designed instead of the normal properties window.



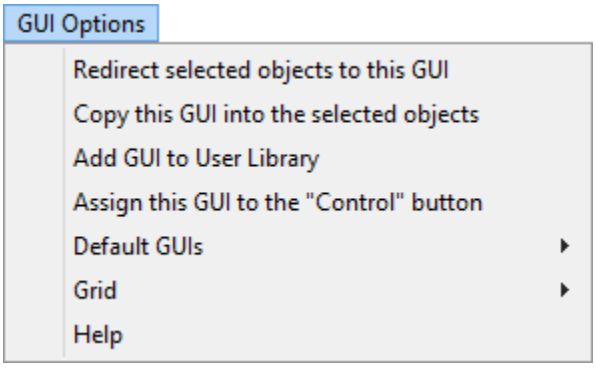
Make some changes to the max speed, acceleration and deceleration values, then press the OK button to apply those changes. Open the window again to verify that your changes were applied correctly.

Graphical User Interfaces Reference

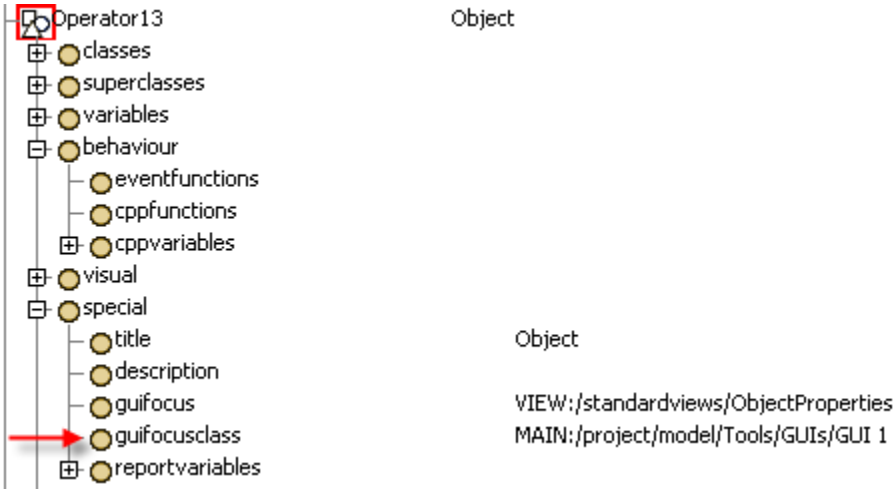
For information on using the Graphical User Interface, see the Example page.

GUI Builder Menu

The GUI builder's edit menu allows you to do several operations with the GUI once you have created it.

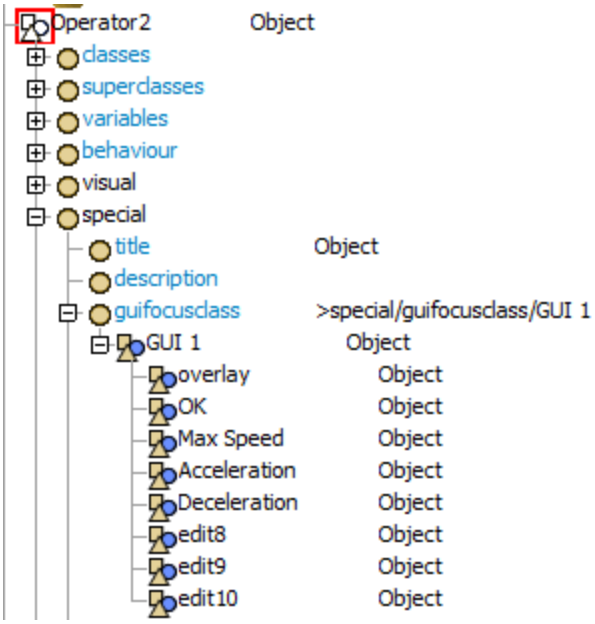


Redirect Selected Objects to this GUI - This option will redirect the guifocusclass attribute of every selected object in the model to point to this GUI. To illustrate this, explore the model tree, and find the operator that you redirected to the GUI you created. Expand its attribute tree and the expand the node named "special". Inside of special there should be a node named guifocusclass. This node's text specifies a path to a window that will be opened when the object is double clicked on.



Notice that the path for this attribute is "MAIN:/project/model/Tools/GUIs/GUI 1". This is where our GUI is stored. When we selected the redirect option, it changed this path from the Operator's normal Properties page to our page.

Copy this GUI into Selected Objects - This option will create a complete copy of the GUI and store it inside of each selected object. To show exactly how this works, let's do it for the GUI we created in our example. Go back into the toolbox and open the GUI builder and GUI canvas for the GUI you've built. Then, with our original operator still selected, select the menu option: Edit > Copy this GUI into Selected Objects. Now go back into the model tree view and look at what was done with the operator's guifocusclass attribute.



Now the guifocusclass attribute has been changed to ">special/guifocusclass/GUI 1". Also, a copy of the entire GUI that we created has been copied into the guifocusclass attribute.

Although you will not need to use the "Copy this GUI into Selected Objects" option if you are just using this GUI for this model, it is useful for portability purposes. Once you have created this GUI and copied it into the object, you can add the object to a user library, and then drag it into other models, and the GUI will be created with it.

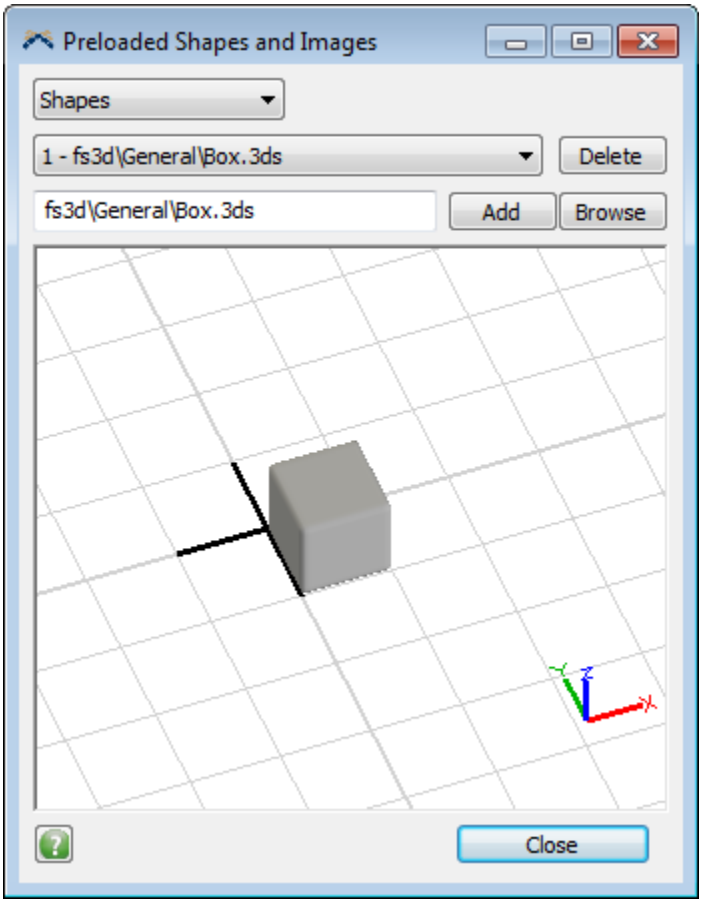
Add GUI to User Library - This option will add the GUI to the selected user library in the library icon grid. For more information on user libraries, refer to the user library documentation.

Assign this GUI to the "Control" button - This option causes the "Control" button on FlexSim's main toolbar to open this GUI when it is pressed. This button is called the Model Control GUI button. Its exclusive purpose is to allow model builders to define custom GUIs for controlling models and their properties without having to change FlexSim's view tree. If this button is no longer available, you may add it to your user toolbar through the Global Preferences Window.

Default GUIs - This sub-menu lets you also edit how other buttons on FlexSim's main toolbar operates. The 3D View button on the toolbar can be customized to open custom GUIs that you have created. By selecting "Make this GUI the Default 3D View GUI" you can cause the 3D View buttons on the main toolbar to open your GUI. Select "Reset Default 3D View GUI to Original" to cause the 3D View buttons to revert back to their original GUIs. Note that making a custom GUI for the 3D View buttons only applies to the model you are editing. When you open another model, the 3D View buttons will reset to the original default view.

Grid - This sub-menu lets you set whether the GUI canvas snaps to grid and what that grid size is.

Media Files



Media Files can be accessed from the Tools menu > **Media Files**.

This editor allows you to add 3D shapes and images to be pre-loaded to the model, as well as get the path string and index for shapes/images that are already loaded. Alternatively, you can import 3D shapes or textures (images) through an object's Properties window or in the Quick Properties. Shapes assigned through these methods will automatically be imported into the model as a preloaded shape.

The number next to each option in the drop-down list can also be used if you are writing code that references a texture or shape index using commands like:

```
setobjectshapeindex(obj, 1);
```

```
setobjecttextureindex(obj, 1);
```

Shape indexes may change as shapes are added/removed from the model. To get the current shape index for a 3D shape use:

```
getshapeindex("fs3d\\General\\Box.3ds");
```

For more information on 3D Media in FlexSim, see the 3D Media section of this help manual.

Shapes - Switches the window between displaying 3D shapes and images.

Shape/Image Combobox - Displays all the currently loaded shapes or images.

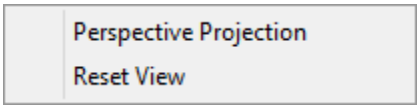
Delete - Delete's the currently selected shape or image. If the shape or image is in use it will not be able to be deleted.

Path - Displays the path to the 3D shape. This is also the "name" of the shape when using commands like getshapeindex().

Add - Adds the current shape or image as a preload.

Browse - Allows you to browse Windows and select an image or shape to import. This will only display the path to the shape/image in the Path field and display it in the view. You must press the Add button to add the shape or image as a preload.

2D/3D View - Displays the currently selected shape in 3D. Right-click on the view to display the following menu:

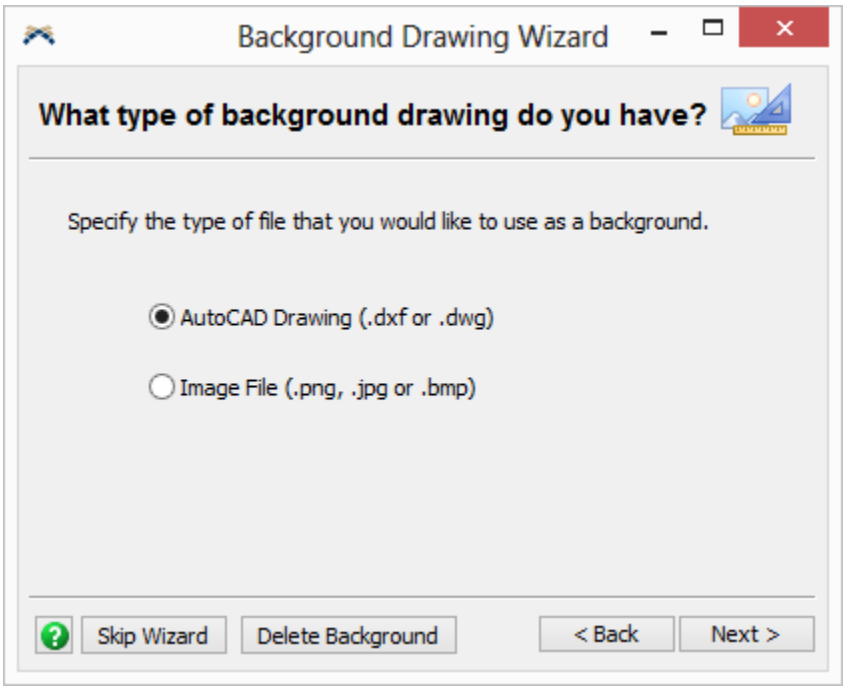


Perspective Projection - Toggles the window to be orthographic or perspective (see Perspective View for more information).

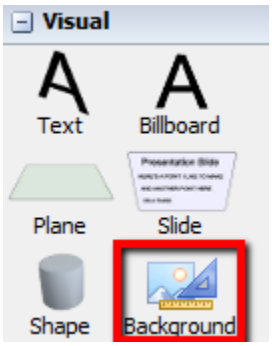
Reset View - Resets the view's position and rotation to 0, 0, 0.

Close - Closes the window.

Model Background


Background Drawing Wizard
What type of background drawing do you have?
Specify the type of file that you would like to use as a background.
AutoCAD Drawing (.dxf or .dwg)
Image File (.png, .jpg or .bmp)
Skip Wizard
Delete Background
< Back
Next >

The Model Background can be accessed from the Tools menu > **Model Background** or by dragging and dropping a Background object from the Visual section of the Library Icon Grid.

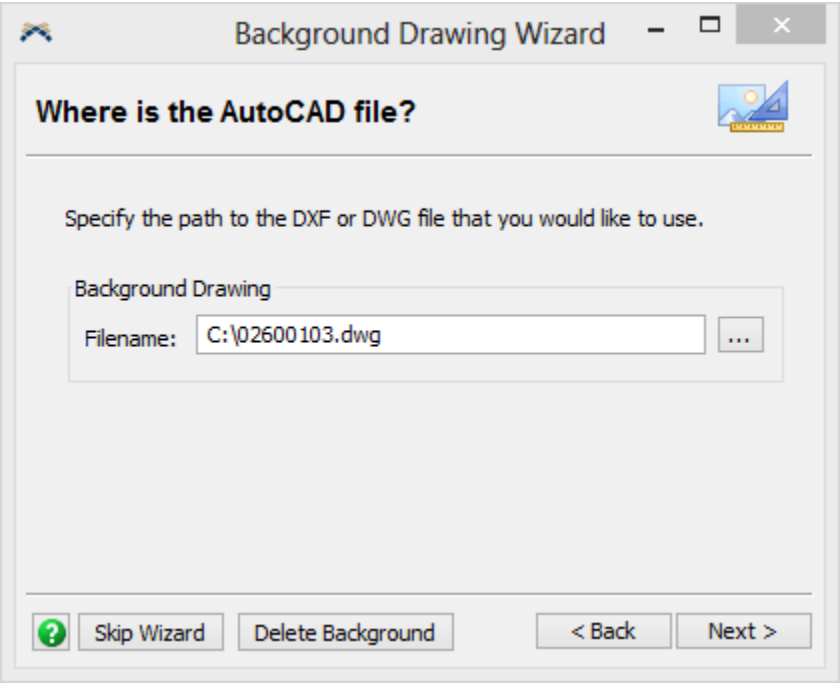


Note: Dragging and dropping a background object into a model already containing a background will open the previously created background's properties window.

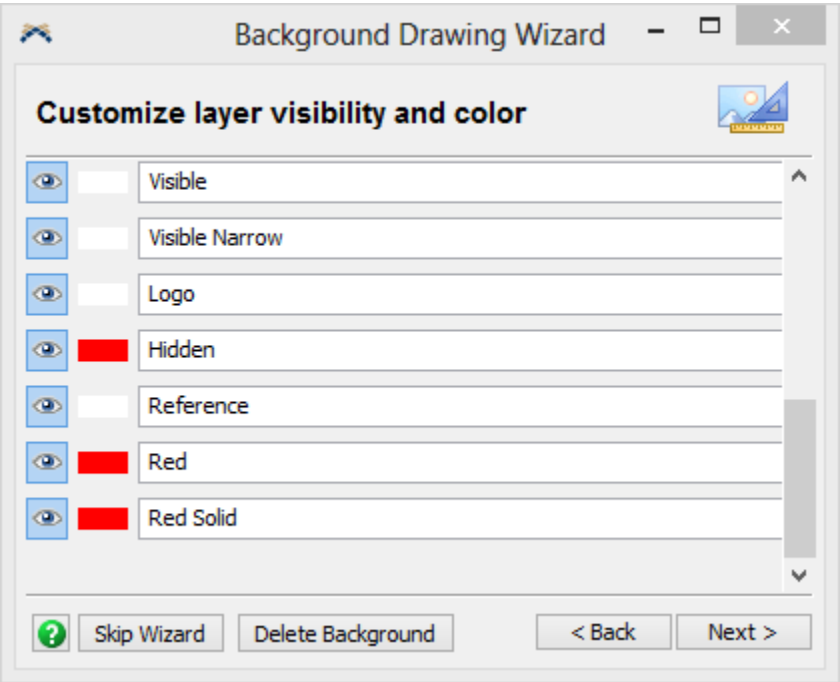
The Model Background is just a Visual Tool object that displays either and image or a dxf/dwg file as the "floor" of the model.

Select your desired format (AutoCAD Drawing or Image) and browse for your desired file.

DWG Files



DWG files contain multiple layers which may be customized within FlexSim. Once selecting your desired file, click the Next button to view available layers.



You can turn layers on an off by clicking on the button. Next to the eye is a color well where you can define that layer's color.

DXF Files

Select a DXF file just as you would a DWG. However, there is no Layer Visibility and Color page to edit.

Image Files

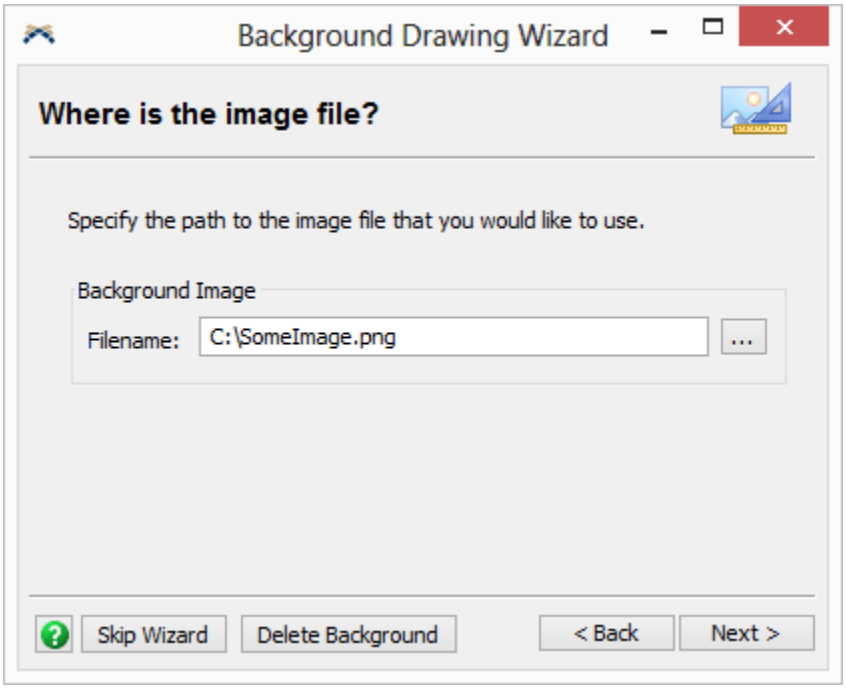
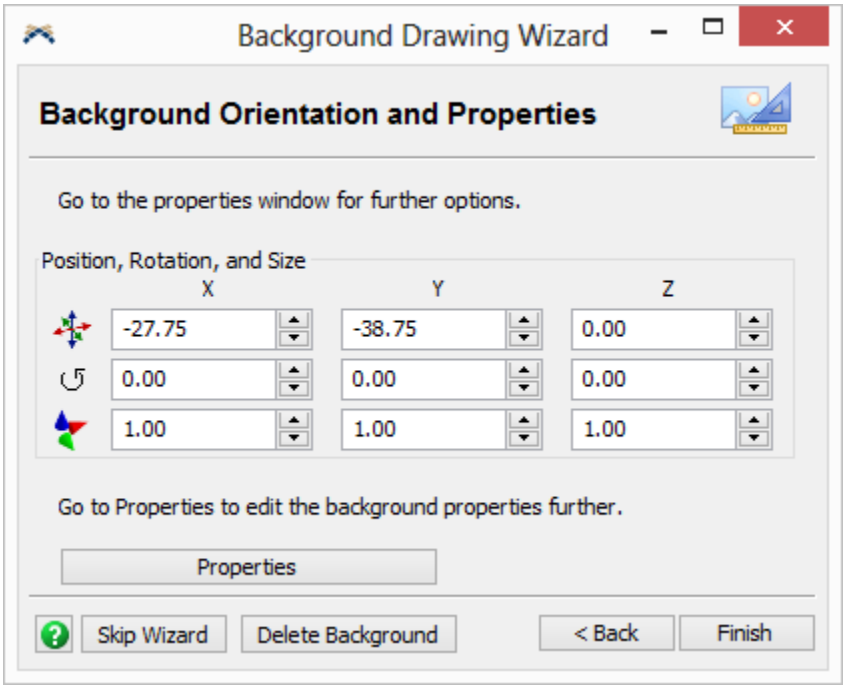


Image files may be png, jpg or bmp files.

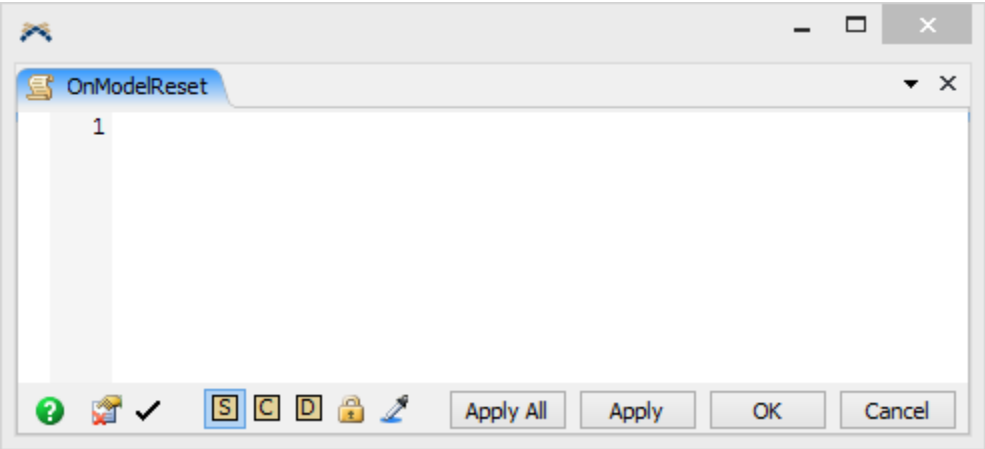
Setting the Position, Size and Rotation



Some files may not import with the correct position, size or rotation. This can easily be fixed through this final page.

DWG and DXF files both have length units associated with the file. When you import these into FlexSim, you want those units to match your model's units. Let's say you are importing a DWG or DXF file that is the layout of your company's factor floor. If your FlexSim model units are set to meters and the DWG or DXF file you were given was set to feet, the image will be scaled too large. This is because FlexSim imports the model as a 1-1 unit ratio. So 1 foot is equal to 1 meter. To scale your DWG or DXF file to the correct size, for this example, you would set the x, y and z scale to 0.3048.

Model Triggers



Model Triggers can be accessed through the Tools menu > **Model Triggers**.

Model triggers allow you to execute code at different points between model runs. The following triggers are available:

On Model Open - This trigger is fired when the model is opened from the file.

On Reset - This trigger is fired when the model is reset.

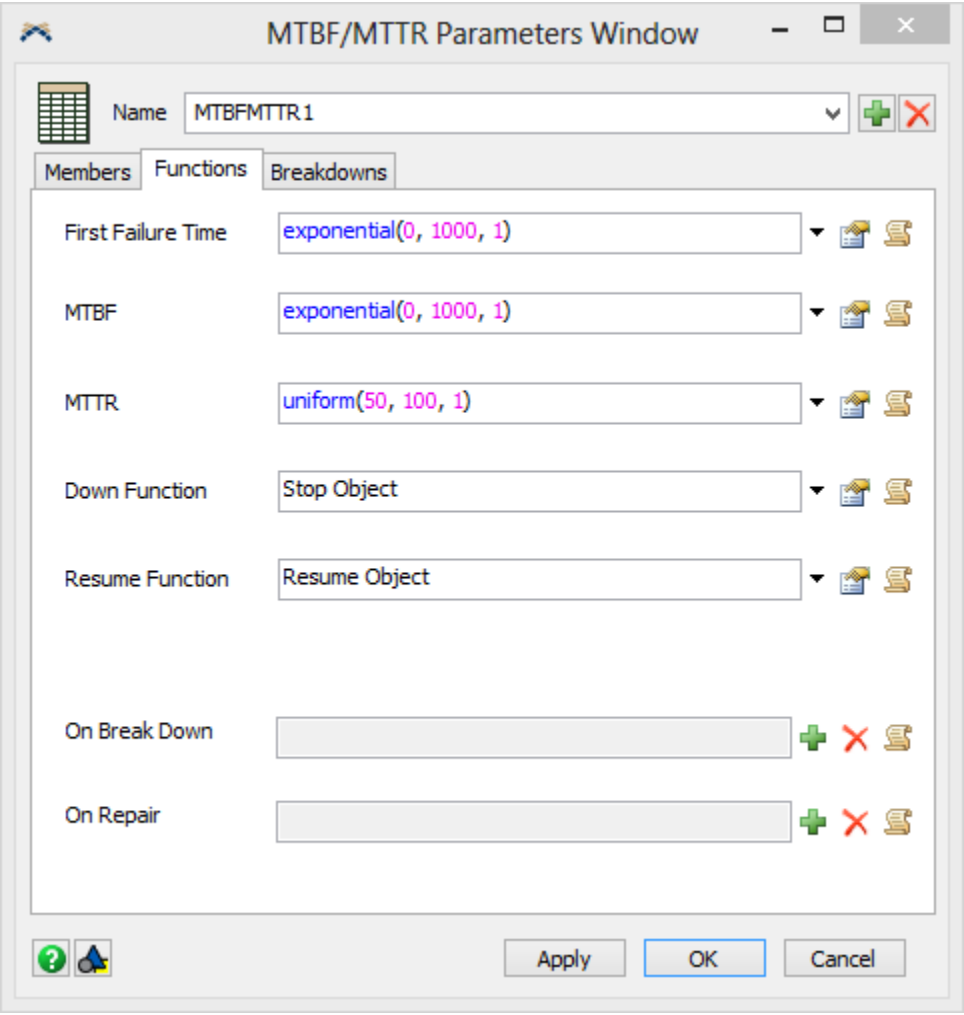
On Run Start - This trigger is fired whenever the model changes from a stopped or paused state to a running state.

On Run Stop - This trigger is fired whenever the model changes from a running state to a stopped or paused state.

On Post-Compile - This trigger is fired after the completion of a compile (see Build menu and When to Compile).

For more available triggers, see the triggers section of the Model Libraries Node page.

MTBF/MTTR



MTBF/MTTR objects are accessed from the Tools menu > **MTBF MTTR**.

MTBF/MTTR objects are used to set random breakdown and recovery times for groups of objects in the model. Each MTBF/MTTR object can have any number of object members and each object can be controlled by more than one MTBF/MTTR object. The MTBF/MTTR object allows you to also specify what state the objects will go into when they go down and what behaviour they should perform. A model may contain any number of MTBF MTTR objects.

Though similar to the Time Table, the MTBF/MTTR object uses picklists to determine dynamically when the connected members will break down and how long they will be broken down for. You can also specify more specific information about the breakdowns. This includes specifying if all connected members will breakdown together, or if the breakdown times will be individually calculated for each object (firing the First Failure Time, MTBF and MTTR picklists once per object). If you only want the connecrted members to go down while they are in a specific state, for example a Processor when it is in the *processing* state, this can be specified as well. This means, if the MTBF time comes and the Processor is currently *idle* , then the Processor will not go down until it enters the *processing* state.

Pages

- Members
- Functions
- Breakdowns

Name - The name of the MTBF/MTTR. The combobox has a list of all MTBF/MTTR objects in the model, allowing you to quickly jump to different MTBF/MTTR objects.

- Create a new MTBF/MTTR object.

- Delete the current MTBF/MTTR.

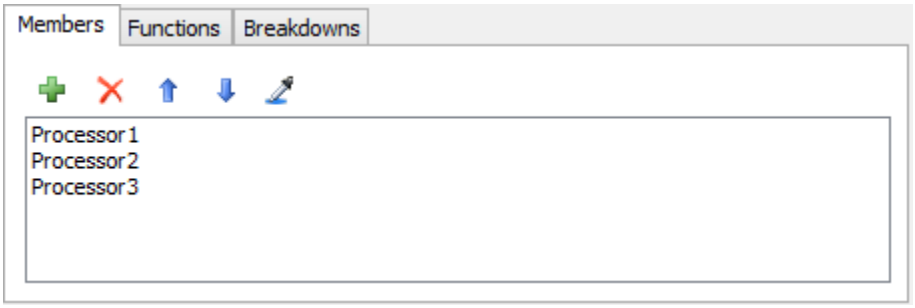
- Adds the MTBF/MTTR to a User Library as either a Draggable Icon or an Auto-Install Component.

Apply - Saves all changes to the MTBF/MTTR.

OK - Saves all changes to the MTBF/MTTR and closes the window.

Cancel - Cancels any unsaved changes made to the MTBF/MTTR and closes the window.

Members Page



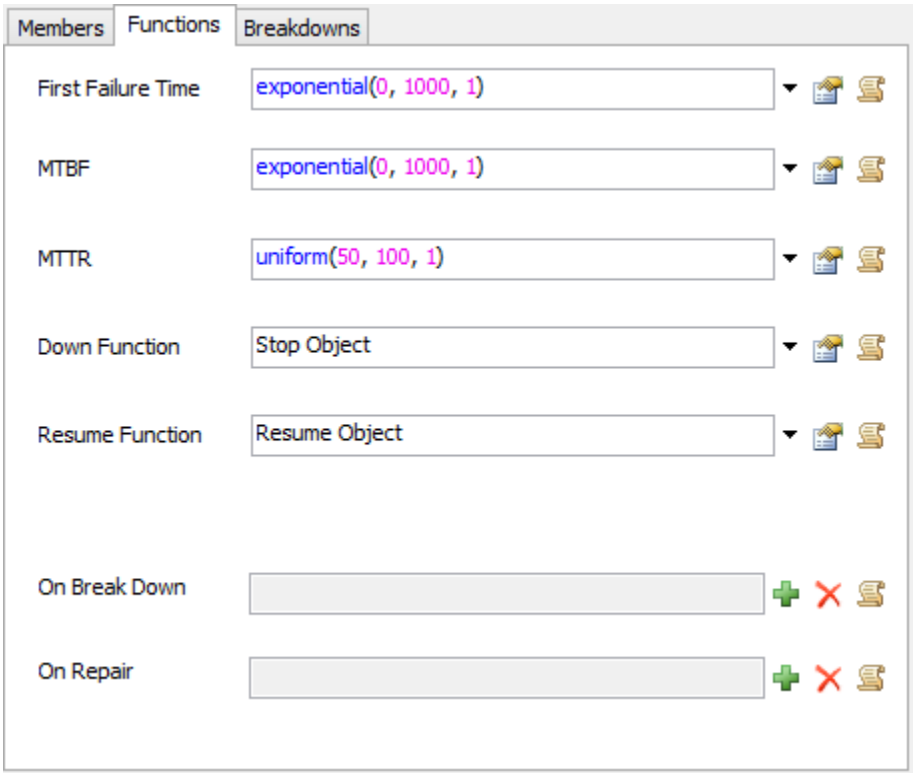
- This will open an object selection GUI where you can select multiple objects in the model.

- Removes the selected member(s) from the list.

- Reorder's members Up or Down in the list.

- Click to enter "Sample" mode, then click on any object in the model to add it as a member.

Functions Page



The following picklists can be fired individually for each object, or for all the objects together depending on the checked state of **Break down members individually** from the Breakdowns Page.

First Failure Time - This picklist returns the time of the first failure. See the time picklist.

MTBF - This picklist returns the Mean Time Between Failure for the objects controlled by this MTBF MTTR object. This function determines how long those objects will run before they go into a broken-down state. The MTBF time is specifically defined as the span between the time that the object resumes from its last down period and the time that it starts its next down period. See the time picklist.

MTTR - This picklist returns the Mean Time To Repair for the objects controlled by this MTBF MTTR object. This function determines how long they will stay in a broken-down state before resuming normal operations again. All of the controlled objects will go back to their original states at the same time. See the time picklist.

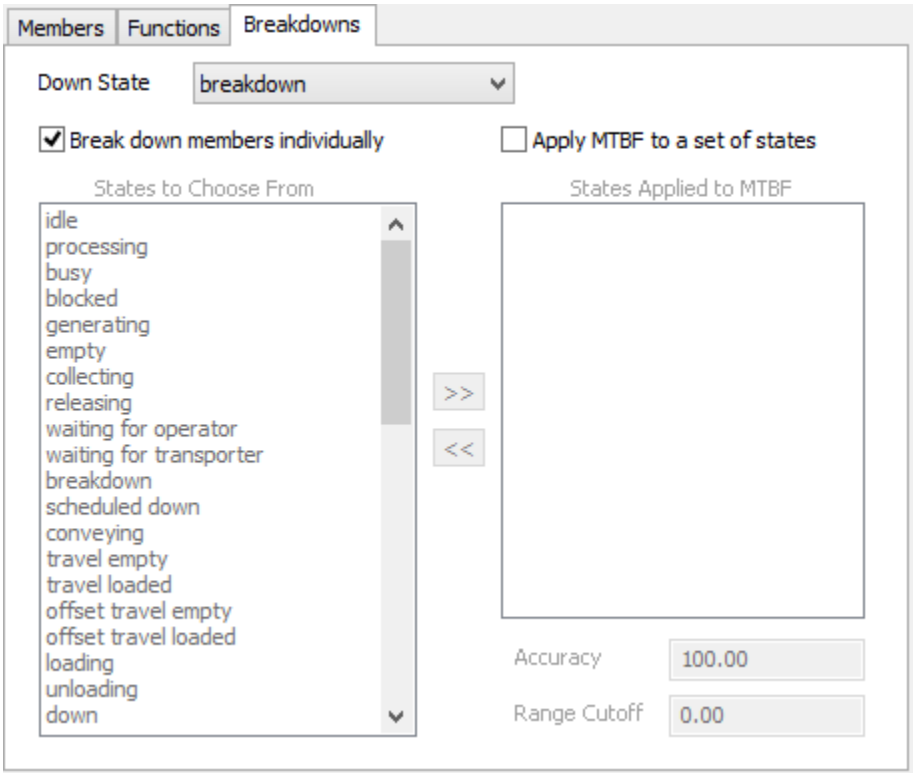
Down Function - This picklist is executed when the objects in the member list go down. It is executed once for each object in the member list. Here is where you specify what to do to stop the object.

Resume Function - The picklist is executed when the objects in the member list resume their operation. It is executed once for each object in the member list. Here is where you specify what to do to resume the object.

OnBreakDown - This picklist is fired immediately after the Down Function, but it is only executed once, instead of once for each object. See Down/Resume Trigger.

OnRepair - This picklist is fired immediately after the Resume Function, but it is only executed once, instead of once for each object. See Down/Resume Trigger.

Breakdowns Page



Down State - This specifies the state that the object will go into when it goes down.

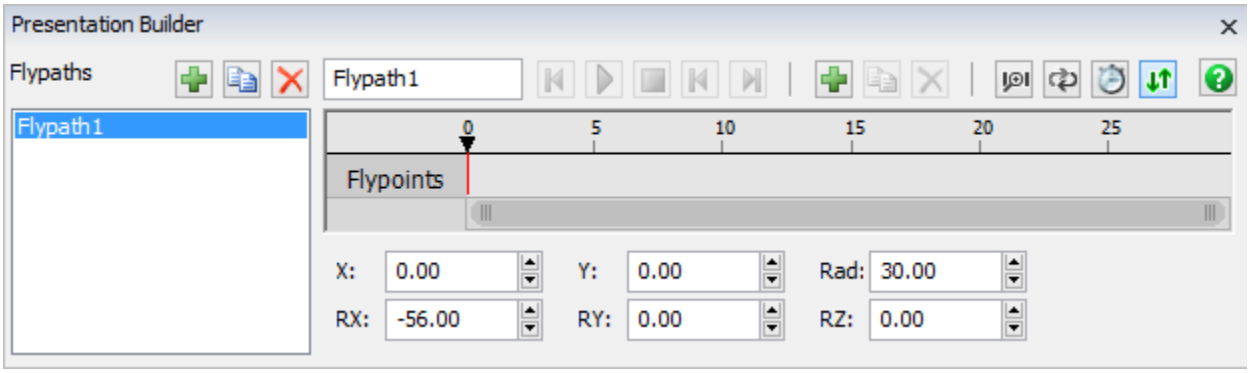
Break down members individually - If this box is checked, the MTBF/MTTR object will create a separate thread of down and resume events for each member object. If it is not checked, all member objects will go down and resume at the same time.

Apply MTBF to a set of states - This box only applies if the MTBF/MTTR breaks down members individually. If it is checked, then the MTBF time will only be applied to a subset of the object's state. For example, if machine break down data only applies for when the machine is actively processing, then you would use this field. If checked, you will add a set of states to the list on the right from the list of possible states on the left.

Accuracy - This field only applies if the MTBF/MTTR uses a subset of the object's states for its MTBF time. Usually this value will be 100, or 100% accuracy. However, if the subset of states represents a very small portion of the total time of the member objects' state times, then the accuracy value can be used to optimize for run speed. For example, if an MTBF is applied to an object's "Waiting for Operator" state, but the object is only in that state 5% of the time, an accuracy value of 100 would cause the MTBF to perform several checks before bringing the object down. If you change the accuracy value to 5, then the MTBF do much fewer checks before bringing the object down.

Range Cutoff - This field only applies if the MTBF/MTTR uses a subset of the object's states for its MTBF time. Usually this value will be 0. However, if the subset of states represents a very small portion of the total time of the member object's state times, then the range cut-off value can be used in conjunction with the accuracy value to improve run speed. This specifies a range within which the MTBF can go ahead and bring the object down. For example, if the next down is scheduled for when the object's subset of states has reached 10000, and the range cutoff is 100, the MTBF will do a check, and if the state subset is above 9900, it will go ahead and bring the object down.

Presentation Builder



The Presentation Builder is accessed from the Tools menu > **Presentation** > **Presentation Builder**.

The Presentation Builder that will assist you in developing a fly-thru presentation of the model. You can create multiple flypaths each with their own set of flypoints. When run, the 3D view will sequentially fly or move to each flypoint in the flypath. When used with the presentation slide option of the visual tool the presentation builder can develop PowerPoint™ style presentations in 3D. If you do not have a 3D view active, the Presentation Builder will be grayed out.

Flypaths are not associated with simulation speed. Flypaths travel in real time by default. That setting can be changed by toggling the button. Flypaths also do not start/stop when the model is started/stopped. The Presentation Builder has its own set of Start and Stop buttons. However, when using the AVI Maker, you can specify one your flypaths to be run while recording your avi file.

Creating a Flypath

Creating a flypath is easy in FlexSim. Once you have the Presentation Builder open, move/rotate the 3D view to the position you would like the flypath to start at. Press the to add a flypoint. This is similar to a keyframe in the Animation Creator. Notice that the Presentation Builder automatically moved the flypath time cursor two seconds ahead of your created flypoint. This makes it easy to add multiple flypoints very quickly. Move to your next desired position/rotation and click the again. Continue this process until you have created a complete flypath.

You can always insert new flypoints by moving the time cursor to any spot on the time line and hitting the . Delete flypoints by selecting them and pressing the or hitting the Delete key.

You can change the zoom of the timeline by either using the mouse scroll wheel or dragging the ends of the bottom scrollbar.

Note when using the scroll wheel: The timeline must be the active view in order to receive mouse scroll events. You may need to click on the timeline to make it the active view if zooming is not occurring.

Updating Fly Points

You can jump to a previously created flypoint by clicking on it, turning the flypoint red. This means it has been selected. The position/rotation fields below will display that flypoints current position. You can edit those fields directly to change the position of your flypoint, or you can reposition the 3D view to update your flypoint.

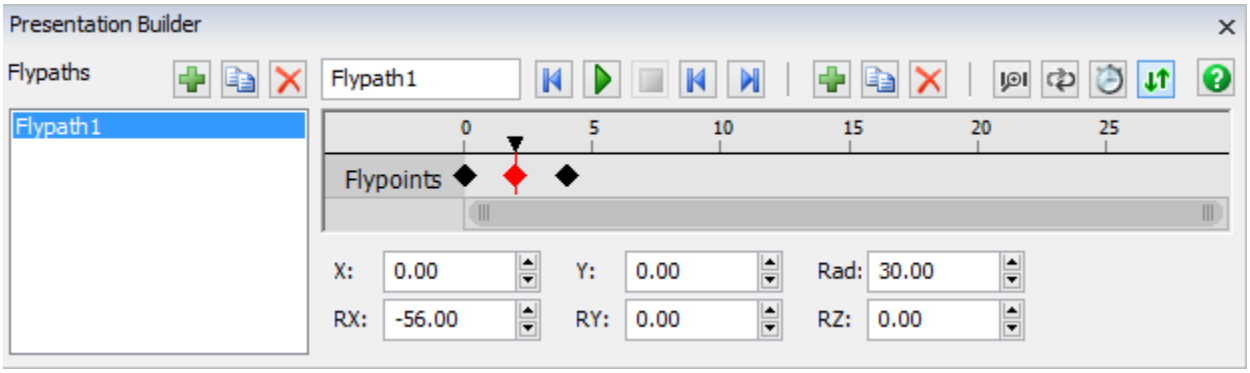
You can move flypoints anywhere along the timeline by clicking and dragging the flypoint to a new position.

Running Fly Paths in the 3D View

Aside from the controls in the Presentation Builder, you can also fly through flypaths by pressing keys on the keyboard. This is useful when you don't want to have the Presentation Builder open. Press one of the numbers: 1-9 to run the associated flypath (plays the flypath from the list at the given rank number). Press

the space bar or the 'N' key, and the view will run the next flypath. Press the 'B' key and the view will go back to the previous flypath.

Reference



Flypaths

- Adds a new flypath.

- Duplicate the selected flypath.

- Removes the selected flypath.

Flypath List - The list of the model's flypaths. Select a flypath to view its properties in the Flypoint Editor.

Flypoint Editor

- Moves the time cursor to the first flypoint in the flypath.

- Runs the flypath from the current time.

- Stop the current flypath.

- Moves the time cursor to the previous or next flypoint.

- Adds a new flypoint at the time cursor's current position.

- Duplicates the selected flypoint.

- Removes the selected flypoint.

- Zooms the timeline in or out to make all flypoints fit on the screen.

- Causes the flypath to loop back to the beginning once it hits the last flypoint.

- If toggled, the flypath will use the model's run speed instead of real time. This can be useful when working with the AVI Maker. By default this option is off and the time displayed in the timeline is in real seconds.

- Sync the 3D View. If toggled, the 3D view will update it's position and rotation as the current time cursor is moved along the timeline.

Timeline - The timeline displays a list of times at the top in seconds. Below that are all the flypoints, or keyframes. You can add any number of flypoints to a flypath.

X - This field specifies the x location of focus point of the camera.

Y - This field specifies the y location of focus point of the camera.

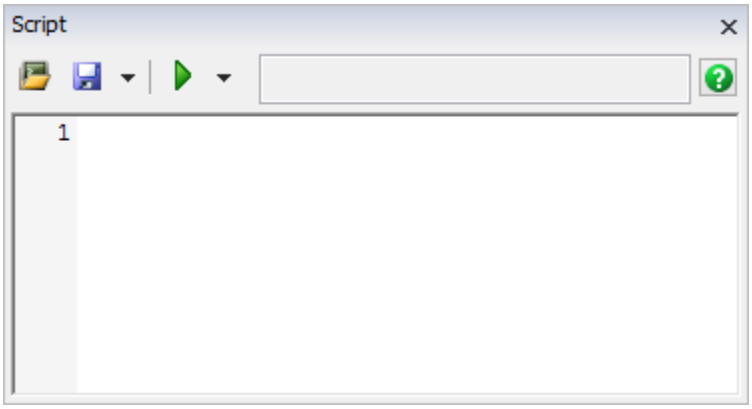
Rad - The radius field specifies the distance the camera is away from its focus or rotation point.

RX - This field specifies the pitch of the camera.

RY - This field specifies the roll of the camera.

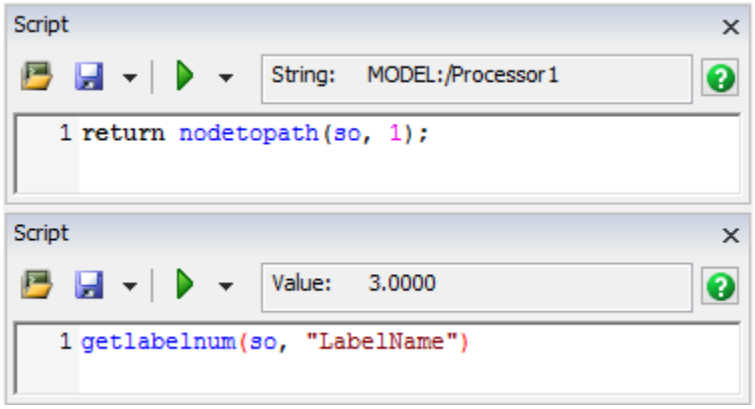
RZ - This field specifies the yaw of the camera.

Script Console



A Script Concsole can be accessed through either the Debug menu > **Script Console**, or through the FlexSim Toolbar.

The script console allows you to execute flexscript commands on the fly without needing to run your model. This can be useful for getting information from your model as well as configuring your model. Type the flexscript code in the main field at the bottom of the window and press the . If your code has a return value, this value will be displayed in the results field. If you are executing a command with a return value like, `getlabelnum` and your code consists of only one line, you can ommit the return and the semi-colon at the end of the line to see the return value in the results field.



If you need to debug your code, you can enter debugging mode by clicking the of the but and select **Debug**. This will place a debug point at line 1 of your code, open it in a Code Editor and execute the code.

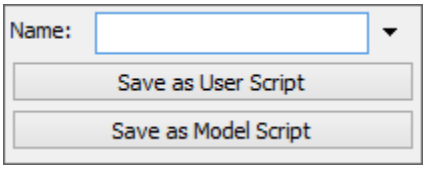
Saving and Loading Scripts

Saving

Code entered into the Script Console can be saved as either a **Model Script** or a **User Script** by pressing .

- Saves the current script. If the script has not yet been saved, opens the Save As popup.

- Opens the Save As popup.



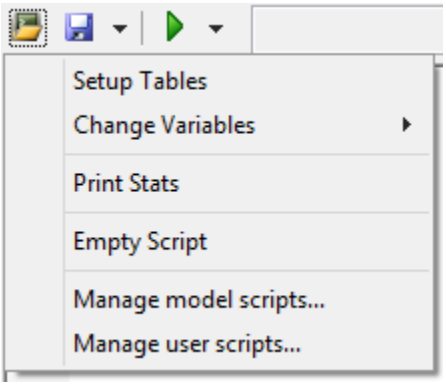
A list of previously saved scripts is available by pressing the .

Save As User Script - These scripts are saved in the user preferences folder (VIEW:/enviornment). These scripts are available for all models while FlexSim is open under your user.

Save As Model Script - These scripts are saved in the model's /Tools/Scripts folder. They are only available for the current model.

Loading

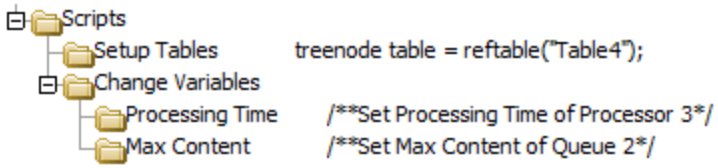
Saved scripts can be loaded by pressing the button. A menu will appear with a list of all Model Scripts, separated by a line, and then all your User Scripts.



Empy Script - Closes any currently open script and clears the code field.

Managing Saved Scripts

As seen in the above image, Change Variables contains a sub-menu of further Model Scripts. You can organize your scripts through any number of sub-menus. To manage your scripts, click the button and select **Manage model scripts...** or **Manage user scripts...**. A Tree Window will appear.

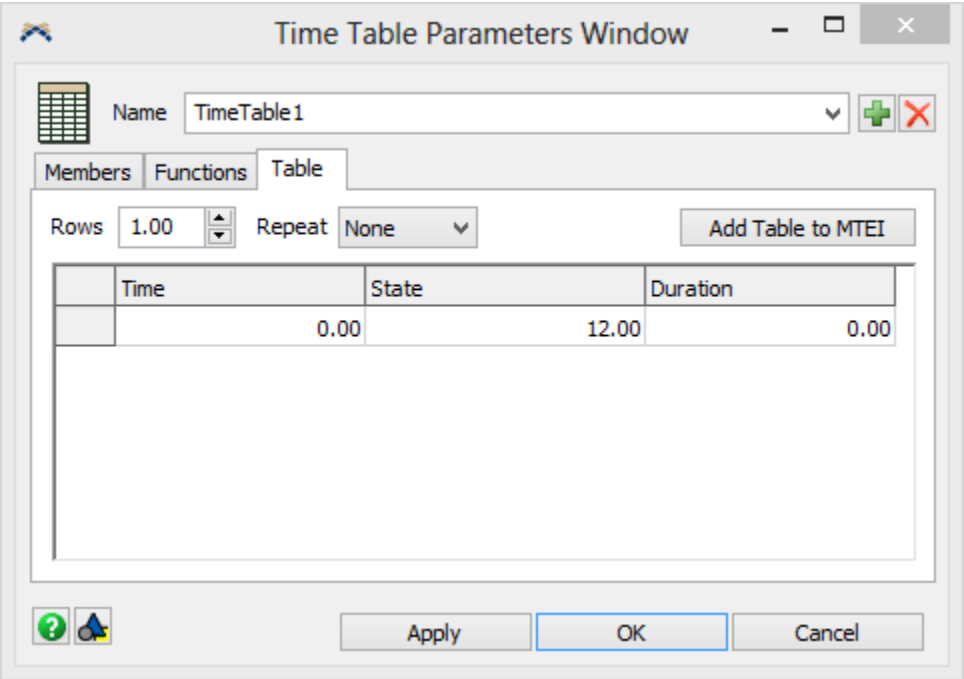


From here, you can organize your scripts by creating sub-nodes and renaming any of your scripts. For more information on using the Tree, see the FlexSim Concepts - Model Tree View or the Tree Structure page.

Time Tables

1. Concepts
2. Reference

Time Tables Concepts



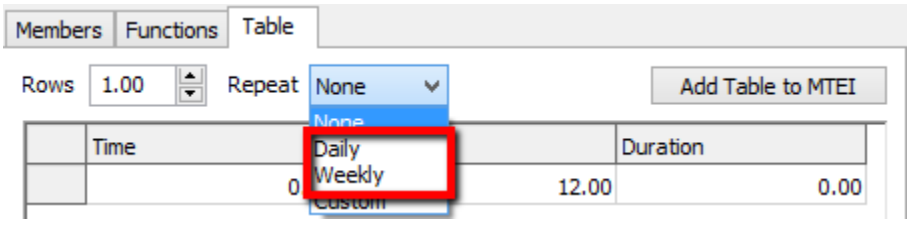
Time Tables are accessed in the Tools menu > **Time Tables**.

Time tables are used to schedule state changes, such as scheduled down-time, for specific objects in the model. Each Time Table may control many objects, and each object may be controlled by many Time Tables. A model may contain any number of Time Tables.

Note on using several down schedules for the same object: If one object has several down schedules, each with its own down state, you may come across problems with the state diagrams of your objects. This is caused because of the nature of the stopobject() and resumeobject() commands. If two entities request the same object to stop, the object does not remember the state for each stop request. For more information, refer to the stopobject() command in the command summary.

Using the Model Start Time

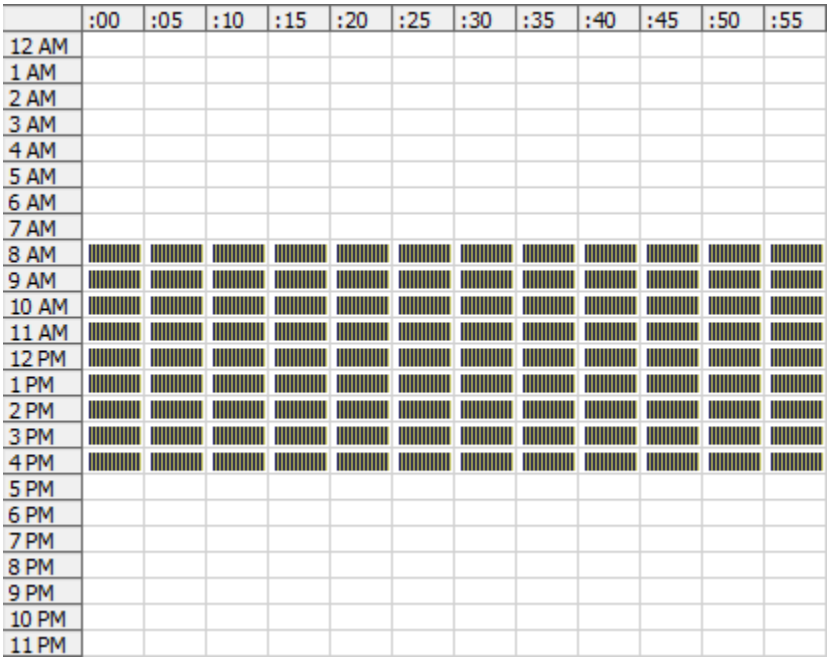
Time Tables can be tied into the Model Start Time as defined in the Model Settings. To tie into the Model Time, the Time Table's Repeat must be set to Weekly or Daily.



Choosing one of these options will give you a graphical view allowing you to specify *Operation Time* and *Down Time* on a daily or weekly basis. Then, based on your Model Start Date and Time, the Time Table will jump into the middle of its table and set its members to be Operational or Down for the given time.

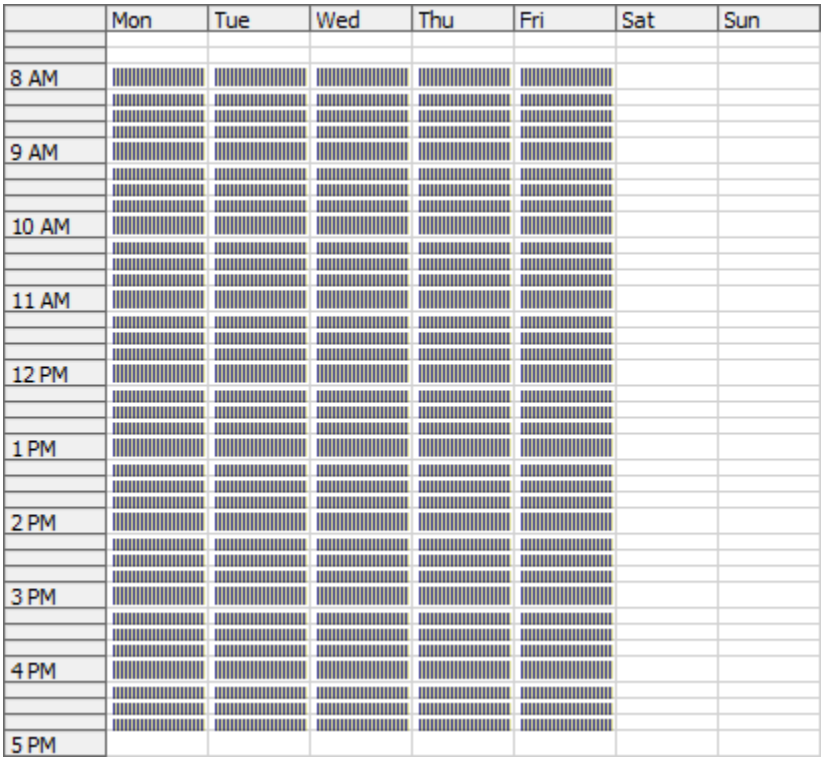
Note: If you want to use the Graphical Time Table to build your Time Table, but you don't want to tie into the Model Start Time, you can set the Time Table's repeat time to *Daily* or *Weekly*, make your necessary changes, hit **Apply**, then set the Repeat time to *Custom*. This will auto fill the numerical table with the correct values associated with the Daily or Weekly table.

Repeat Daily



If you set your Time Table to repeat daily, then the Model Start Date does not come into effect, only the Model Start Time. For example, if your Model Start Time is set to 08:00:00 AM and your Graphical Table looks like the above table, the members of the Time Table will begin the Model in an Operational state, and no functions will be fired. If however, you change the Model Start Time to 07:00:00 AM, when you reset and run your model, the Down functions will fire and the members will begin in a Down state. One hour later (based on the model time units, so 3600 seconds if the model time is set to seconds), the Resume functions will be fired and the members will begin their Operational Time.

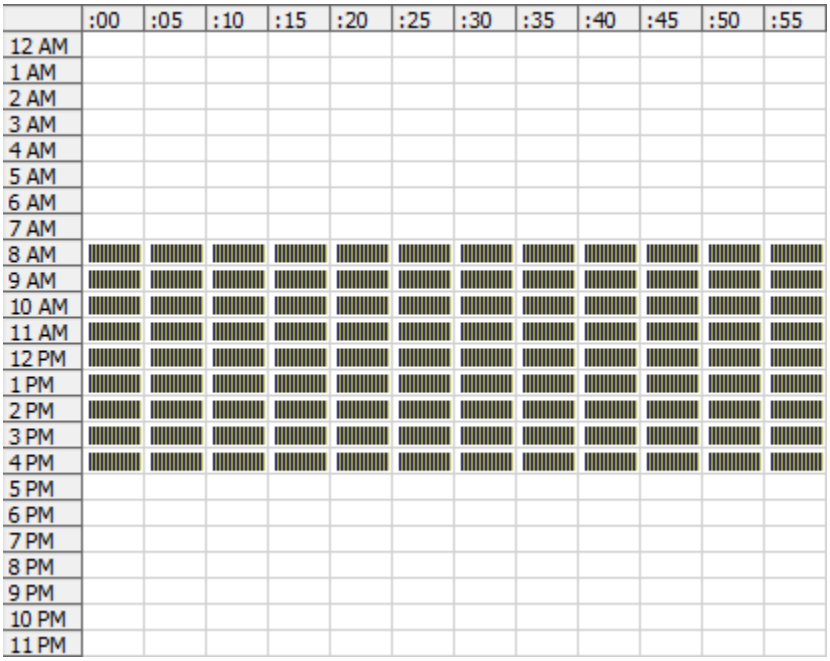
Repeat Weekly



Setting the Time Table to repeat weekly will behave similarly to the Repeat Daily, except that the Time Table will also take into account the day of the week. If the Model Start Time begins on a Tuesday at 08:00:00 AM and our Time Table is set to the above vlues, then the Time Table will skip all of Monday and jump to Tuesday at 08:00:00 AM with the Time Table's members being Operational. When the Model Time

hits Friday at 05:00:00 PM, the members will go Down and remain down until Monday at 08:00:00 AM where the Time Table will start over.

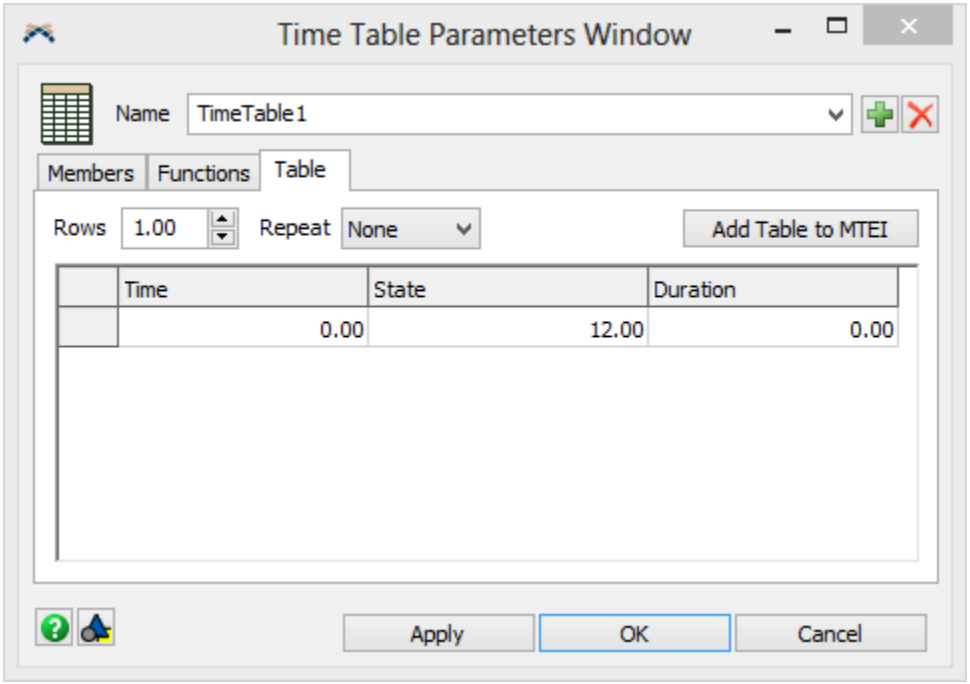
Editing the Graphical Table



Cells marked with ||||| are *Operation Time*. Select a group of cells and press the **Make Operational Time** or **Make Down Time** to specify the member's state for those times. Alternatively, you can add this graphical table to the MTEI and import a table from an Excel workbook. You can also manually delete set times to Down by highlighting one cell and pressing the delete key.

Note on Operational vs Down Time: The Time Table converts this graphical table into numeric values by looking at each cell to see if the cell's string length is great than 0. Thus, having any character inside a cell will tell the Time Table that that time is *Operation Time*.

Time Tables Reference



Pages

- Members
- Functions
- Table

Name - The name of the TimeTable. The combobox has a list of all TimeTables in the model, allowing you to quickly jump to different Time Tables.

- Create a new Time Table object.

- Delete the current Time Table.

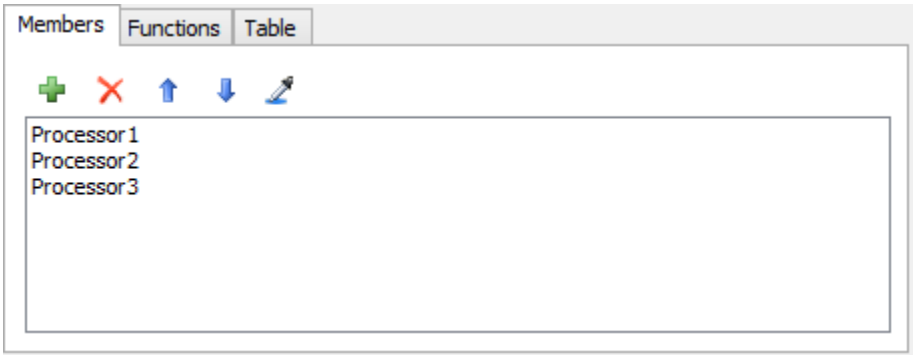
- Adds the Time Table to a User Library as either a Draggable Icon or an Auto-Install Component.

Apply - Saves all changes to the Time Table.

OK - Saves all changes to the Time Table and closes the window.

Cancel - Cancels any unsaved changes made to the Time Table and closes the window.

Members



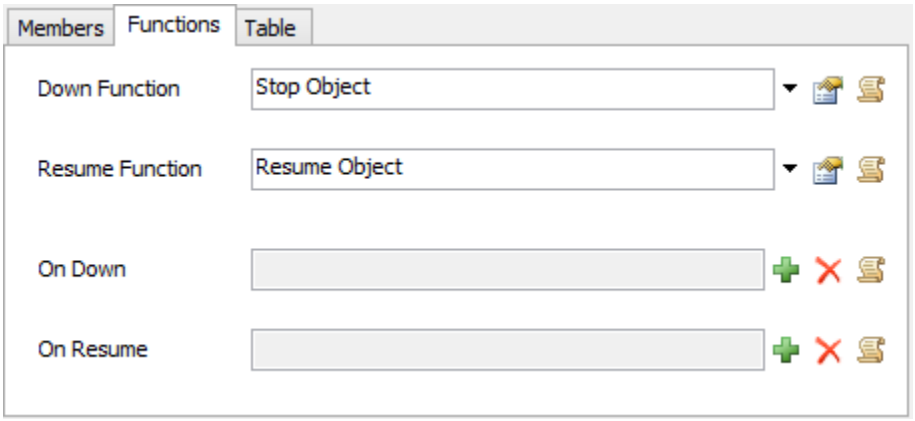
- This will open an object selection GUI where you can select multiple objects in the model.

- Removes the selected member(s) from the list.

- Reorder's members Up or Down in the list.

- Click to enter "Sample" mode, then click on any object in the model to add it as a member.

Functions



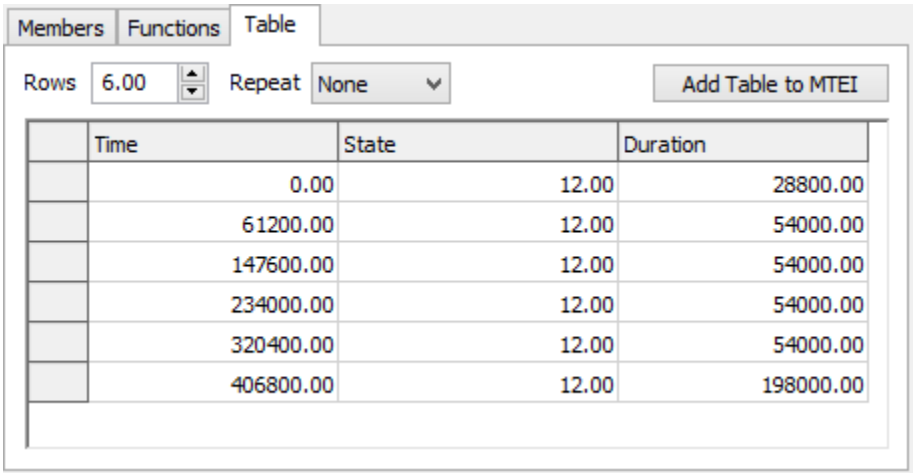
Down Function - This picklist is executed when the objects in the member list go down. It is executed once for each object in the member list. This is where you specify what to do to stop the object.

Resume Function - The picklist is executed when the objects in the member list resume their operation. It is executed once for each object in the member list. This is where you specify what to do to resume the object.

On Down - This picklist is fired immediately after the Down Function has been fired for all objects, but it is only executed once, instead of once for each object in the member list. See Down/Resume Trigger.

On Resume - This picklist is fired immediately after the Resume Function has been fired for all objects, but it is only executed once, instead of once for each object in the member list. See Down/Resume Trigger.

Table



Rows - This is the number of rows in the table.

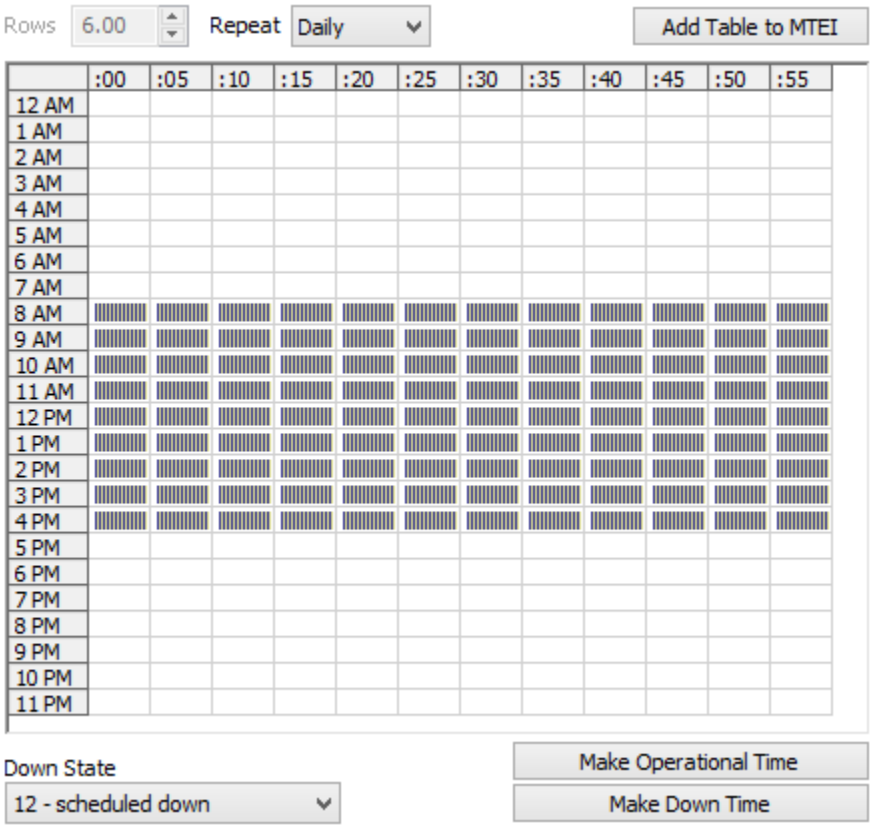
Repeat - This specifies the time from when the first state change begins to when the table is to be repeated. Setting the value to Daily or Weekly will display a Graphical Table. Custom allows you to specify any repeat time. For example, if the first row has a time of 60 and a repeat time of 300, then the first down will occur at time 60, and the table will repeat itself at time 360, 660, 960, etc..

Add Table to MTEI - This button will add the table to the model's multiple table import accessed through the Excel Interface.

Table - Each row records the follwing:

- **Time -** This is the time since the table began that the state change should occur.
- **State -** This is the state that the objects controlled by this table will change into when the time table tells it to go down. If you click on this column, a drop-down box will appear at the top, giving you a list of possible states. Refer to the library objects for more information about what each state means to each object. Refer to the state list for a quick reference of each state's number and macro definition.
- **Duration -** This is how long the objects will stay in the new state before changing back to their original state.

Repeat Daily



For more information on editing the Graphical Table, see the Concepts page.

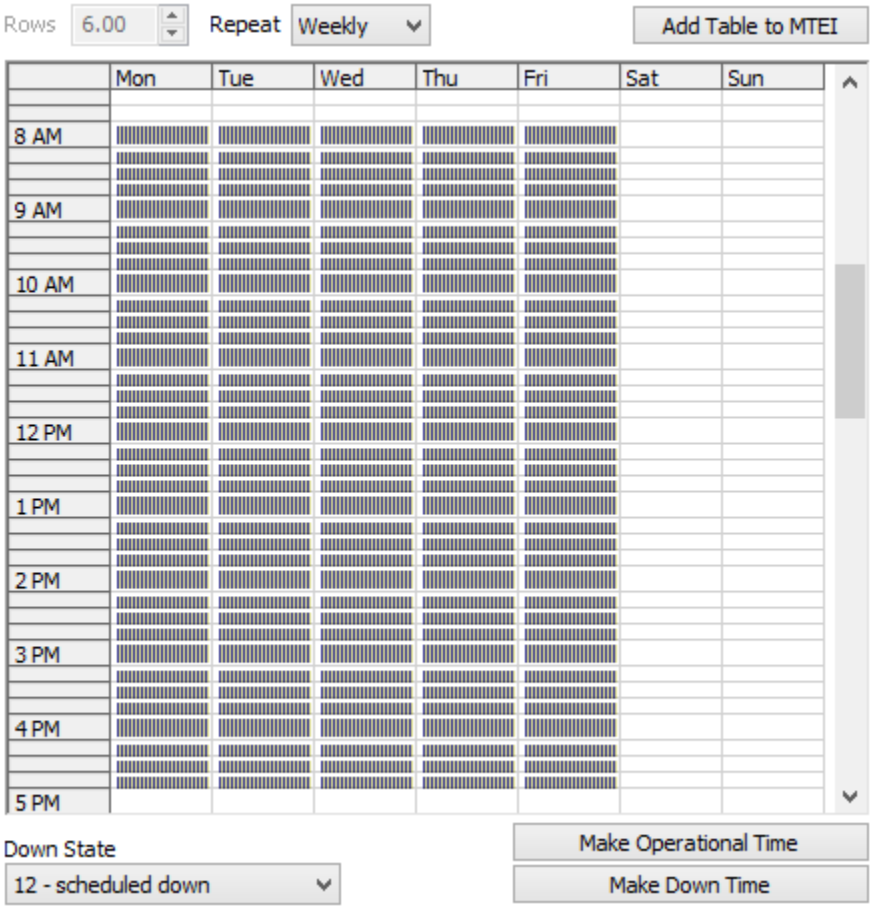
Add Table to MTEI - As explained above, this will add the Graphical Table to the MTEI.

Down State - Specifies what state the members should go to when they enter their *Down Time*.

Make Operational Time - Select a set of cells from the table and press this button to make those times Operational (empty).

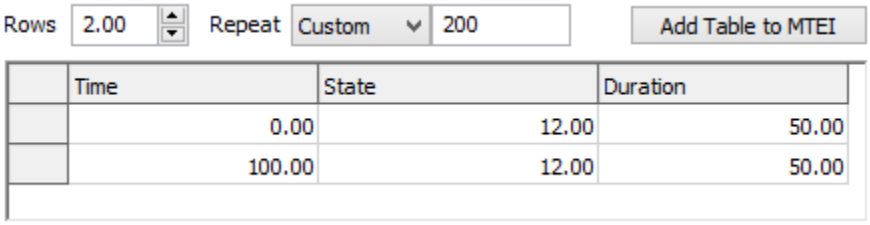
Make Down Time - Select a set of cells from the table and press this button to make those times Down (||||||).

Repeat Weekly



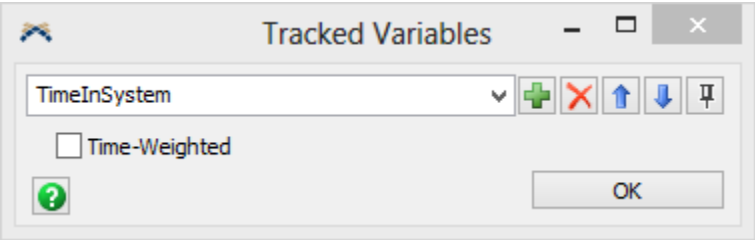
See Repeat Daily above.

Repeat Custom



Repeat Time - Specify the time, in model time units, to repeat the Time Table.

Tracked Variables



Tracked Variables are accessed in the Tools menu > **Tracked Variables**.

Tracked Variables are stored in the model/Tools/TrackedVariables folder. Work In Progress and Time In System are the two default Tracked Variables. When a Flowitem is created, an associated value is added to both of these default variables. These values remain in the model until the model is reset. Custom Tracked Variables may also be added to your model through this window.

- **Work In Progress** - This is a count of all flowitems in the model at any given time. It is incremented when a flowitem is created and decremented when a flowitem is destroyed.
- **Time In System** - This records the time at which an object leaves the model and the total time it spent in the model.

Tracked Variable Drop Down - Shows the current Tracked Variables. Enter text to rename the Tracked Variable.

- Adds a new Tracked Variable.

- Removes the selected Tracked Variable.

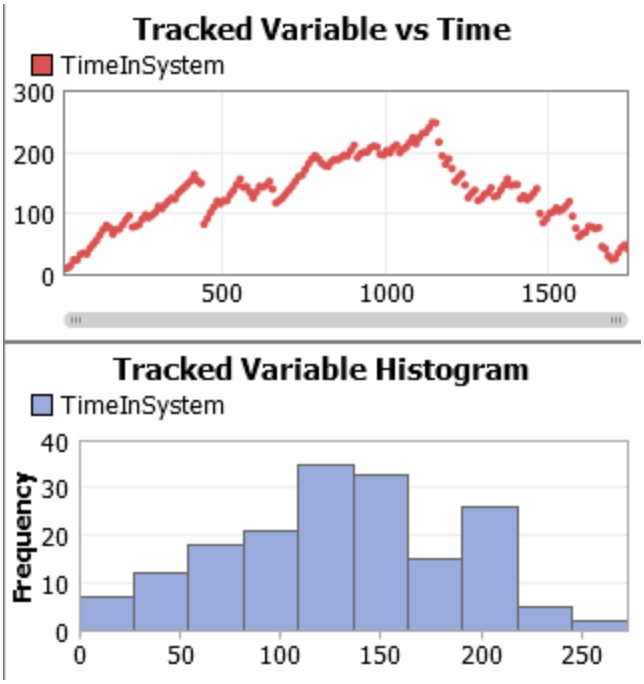
- Reorder's the selected Tracked Variable Up or Down in the list.

- Pins the tracked variable to a Dashboard as either a histogram or line graph.

Time-Weighted - Check this box to have the time that a variable spends at a given value be used as a weight in the Dashboard's histogram. Or in otherwords, the histogram will weight the values based upon the time duration that a variable is at a certain value. If unchecked, Histograms will weight each data point the same no matter how long the tracked variable was at the given value.

Dashboard

Tracked Variables can be displayed in the Dashboard in multiple graphs:



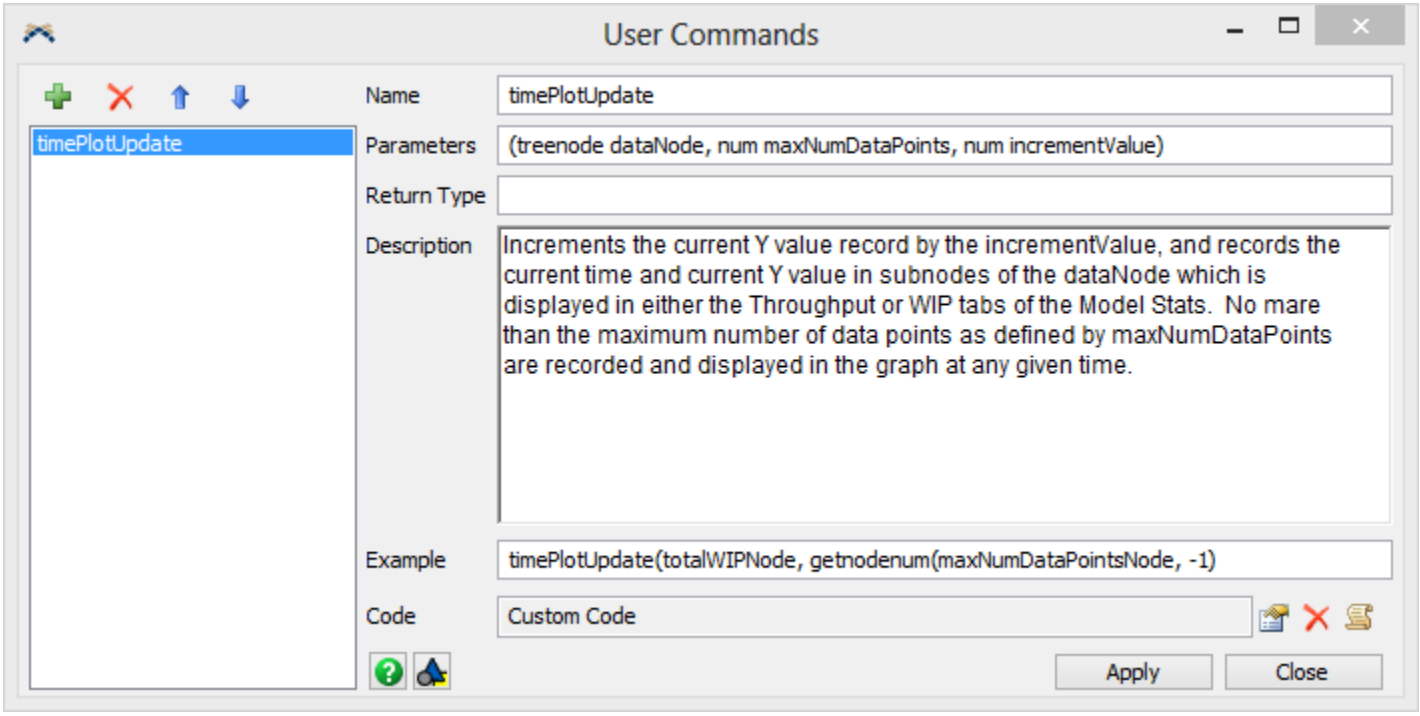
Dynamic Variables

Tracked Variables can be created or customized at any time, using the `settrackedvariable` and `gettrackedvariable` commands. For More information on these commands, see the commands summary page.

User Commands

1. Concepts
2. Reference

User Commands Concepts

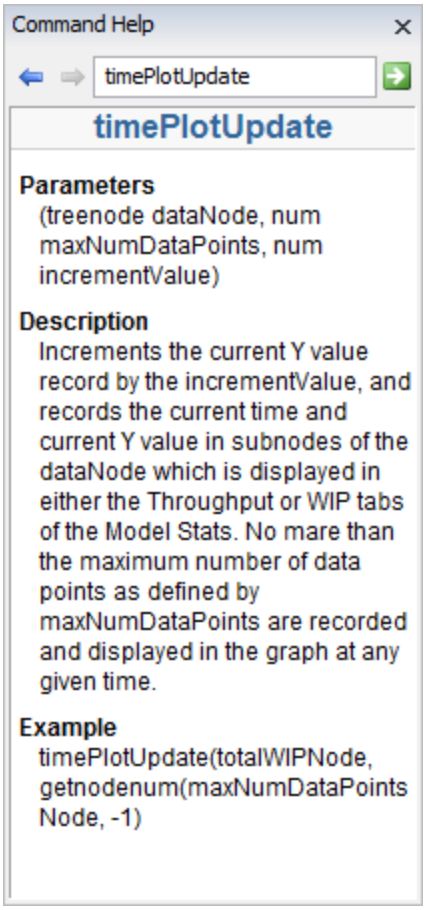

User Commands
timePlotUpdate
Name
timePlotUpdate
Parameters
(treenode dataNode, num maxNumDataPoints, num incrementValue)
Return Type
Description
Increments the current Y value record by the incrementValue, and records the current time and current Y value in subnodes of the dataNode which is displayed in either the Throughput or WIP tabs of the Model Stats. No mare than the maximum number of data points as defined by maxNumDataPoints are recorded and displayed in the graph at any given time.
Example
timePlotUpdate(totalWIPNode, getnodenum(maxNumDataPointsNode, -1)
Code
Custom Code
Apply
Close

User Commands are accessed in the Tools menu > **User Commands**.

The User Commands tool lets you add, delete, and edit custom commands in your model.

Note on calling user commands from C++: When calling a user command from C++, all parameters passed will need to be converted to a number, or else you may get compiler errors. To convert a parameter to a number, use the tonum() command.

Once you have created your command you can call it like any other FlexSim command. It will appear in blue when typing code and it will appear in the Command Summary. You can also hover the mouse over the command and press F1 to display the Command Helper.


Command Help
timePlotUpdate
timePlotUpdate
Parameters
(treenode dataNode, num maxNumDataPoints, num incrementValue)
Description
Increments the current Y value record by the incrementValue, and records the current time and current Y value in subnodes of the dataNode which is displayed in either the Throughput or WIP tabs of the Model Stats. No mare than the maximum number of data points as defined by maxNumDataPoints are recorded and displayed in the graph at any given time.
Example
timePlotUpdate(totalWIPNode, getnodenum(maxNumDataPoints Node, -1)

Accessing Parameters

If your command specifies treenode, number or string data to be passed in as parameters, you can access those parameters through the following commands:

```
parnode(num)
```

```
parval(num)
```

```
parstr(num)
```

Where num is the rank from the parameter list. For example, if the parameters was specified as:

```
(treenode dataNode, num maxNumDataPoints, num incrementValue)
```

You would access those values in your code by:

```
treenode dataNode = parnode(1);
```

```
double maxNumDataPoints = parval(2);
```

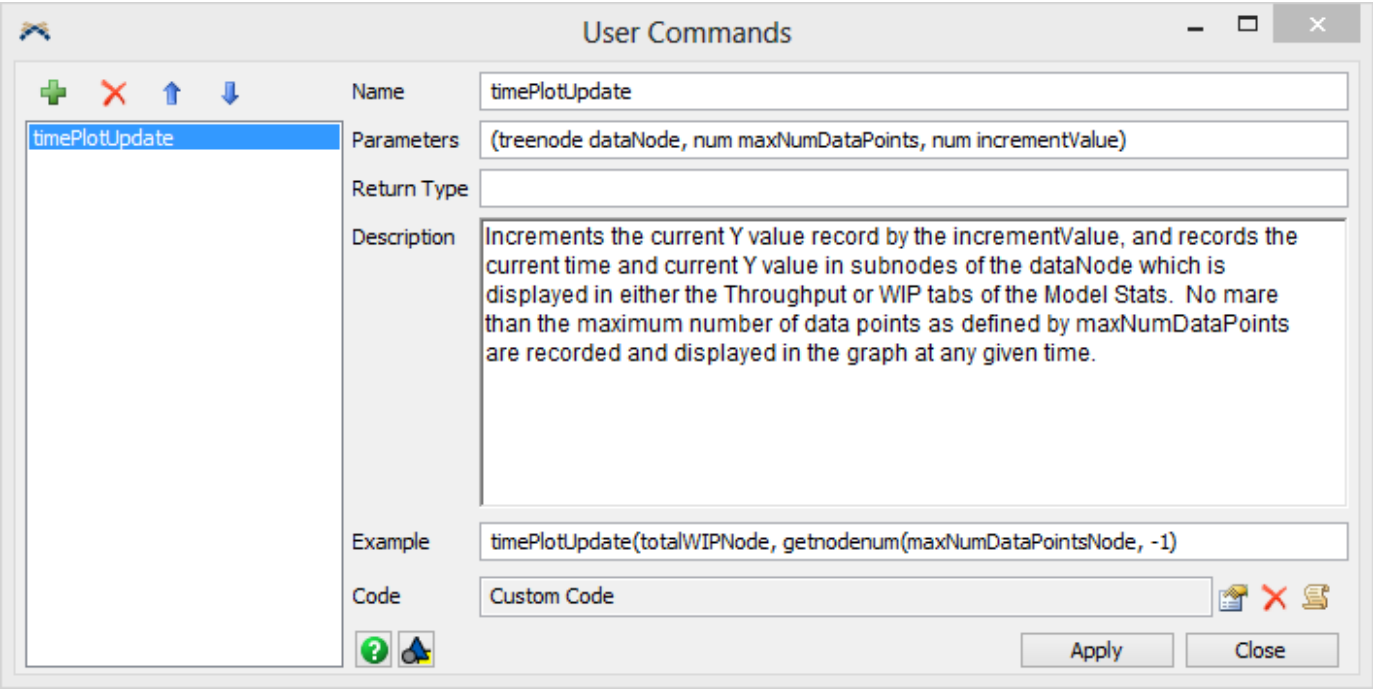
```
int incrementValue = parval(3);
```

Note on Parameters and Return Type: The values specified here will be used by the FlexScript parser. The number of parameters and their datatypes will give FlexScript warnings if not passed in as documented in the Parameters section here. If the return type is incorrecly used in code, it will return a FlexScript error and not build correctly. You can specify the parameters list as (...) and leave the return type blank for the FlexScript parser to ignore them. For examples of parameter lists and return types, view the Command Documentation.

Description

When you open the command documentation from the menu Help > Commands, the user commands will be shown in the command documentation in addition to all of the built-in FlexScript commands. The description field shown above will be displayed as html so you can use html markup for formatting, such as <b> and <\b> for bold. Because the description allows for html markup, some standard characters do not work properly, such as <, >, &, and new line characters. To display these types of characters, you can use special markup codes, such as ">" (without the quotes) for a > symbol and <br/> for a line break. If any of the commands' descriptions container invalid html markup, FlexSim may throw an exception when you try to open the Users Manual and your commands will not be properly documented.

User Commands Reference


User Commands
timePlotUpdate
Name
timePlotUpdate
Parameters
(treenode dataNode, num maxNumDataPoints, num incrementValue)
Return Type
Description
Increments the current Y value record by the incrementValue, and records the current time and current Y value in subnodes of the dataNode which is displayed in either the Throughput or WIP tabs of the Model Stats. No mare than the maximum number of data points as defined by maxNumDataPoints are recorded and displayed in the graph at any given time.
Example
timePlotUpdate(totalWIPNode, getnodenum(maxNumDataPointsNode, -1)
Code
Custom Code
Apply
Close

- Adds a new blank User Command.

- Removes the selected User Command.

- Reorder's User Commands Up or Down in the list.

User Commands List - Displays a list of all User Commands. Select a command from the list to edit it.

Name - The name of the User Command. In writing code, this name is the name of the command, ie getlabelnum().

Parameters - Specifies the parameters that can be passed into the command. Each parameter should be separated by a comma. The following are valid parameter types and their associated Flexscript accessor commands:

- **node/treenode -** In Flexscript, use parnode(paramNum) to get the value of the passed in node.
- **obj -** Same as node.
- **num -** In Flexscript, use parval(paramNum) to get value of the passed in double or int.
- **str -** In Flexscript, use parstr(paramNum) to get the value of the string.

Return Type - Specifies the value the command will return. See the above parameters for value return types.

Description - The description displays in the Command Summary.

Example - The example displays in the Command Summary

Code - Here is where you specify the Flexscript, C++ or DLL code for the command.

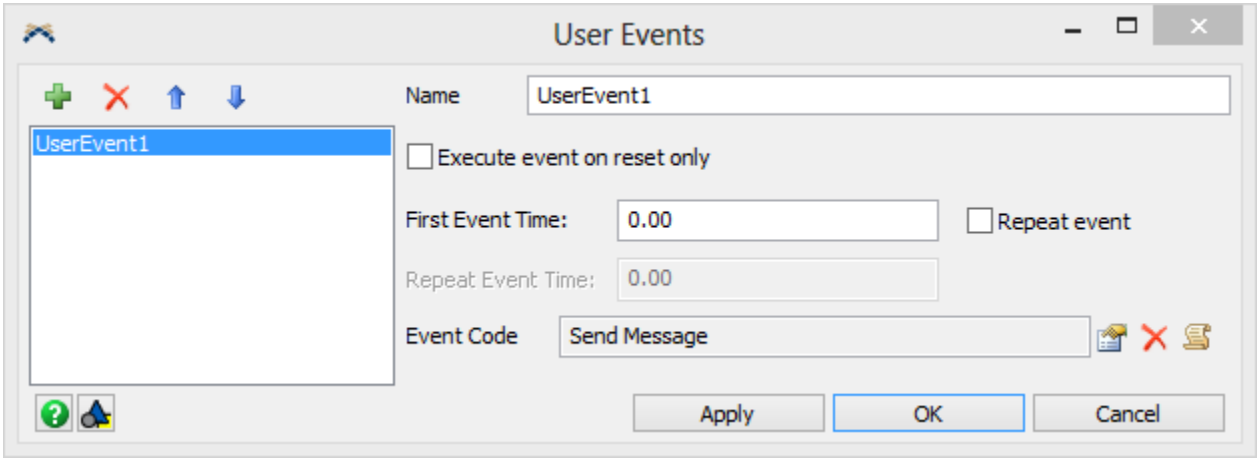
- Displays this help page.

- Adds the User Command or All User Commands to a User Library as either a Draggable Icon or an Auto-Install Component.

Apply - Saves all changes to the User Command.

Close - Closes the User Command window without saving changes.

User Events



User Events are accessed from the Tools menu > **User Events**. User Events are FlexScript functions that execute at set times during the model's run, but are not connected with any specific, visible object. They are created and stored in model's */Tools/UserEvents* folder. A model can have any number of user events.

- Adds a new blank User Event.

- Removes the selected User Event.

- Reorder's User Events Up or Down in the list.

User Events List - Displays all of the Model's User Events. Changing from one User Event to another will apply any changes made to the previous User Event before displaying the newly selected User Event.

Name - This is the name of the user event. This is purefly for the modeler's convenience and has no affect on the model. It should, however, be descriptive of what the user event does.

Execute event on reset only - If this box is checked the event will only be executed when the reset button is pressed.

First Event Time - This is the time in model units that the User Event will occur.

Repeat Event - If this box is checked, as soon as the user event executes, it begins counting towards the next execution time as defined by the Repeat Event Time.

Repeat Event Time - If the Repeat Event box is checked, this field will be enabled. Once the first User Event executes, the User Event will repeat in regular intervals defined by this time (in model units).

Event Code - This is where the FlexScript code for the event is written. Any valid FlexScript statements can be used in this picklist.

- Adds the User Event to a User Library as either a Draggable Icon or an Auto-Install Component.

Apply - Saves all changes to the User Event.

OK - Saves all changes to the User Event and closes the window.

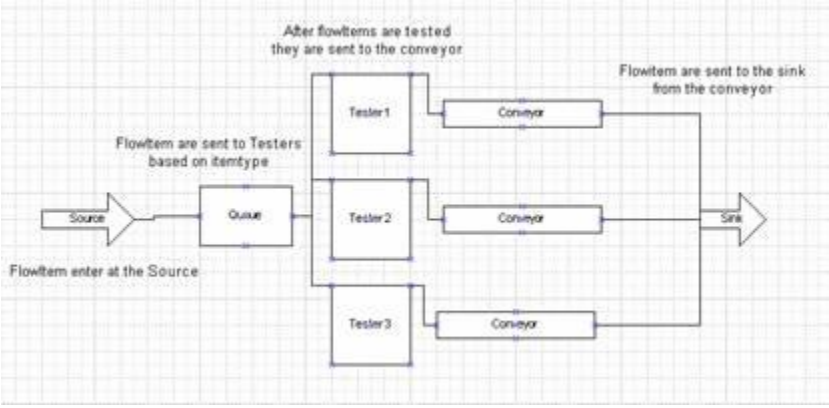
Cancel - Cancels any changes made to the User Event and closes the window.

Microsoft Visio™ Importer

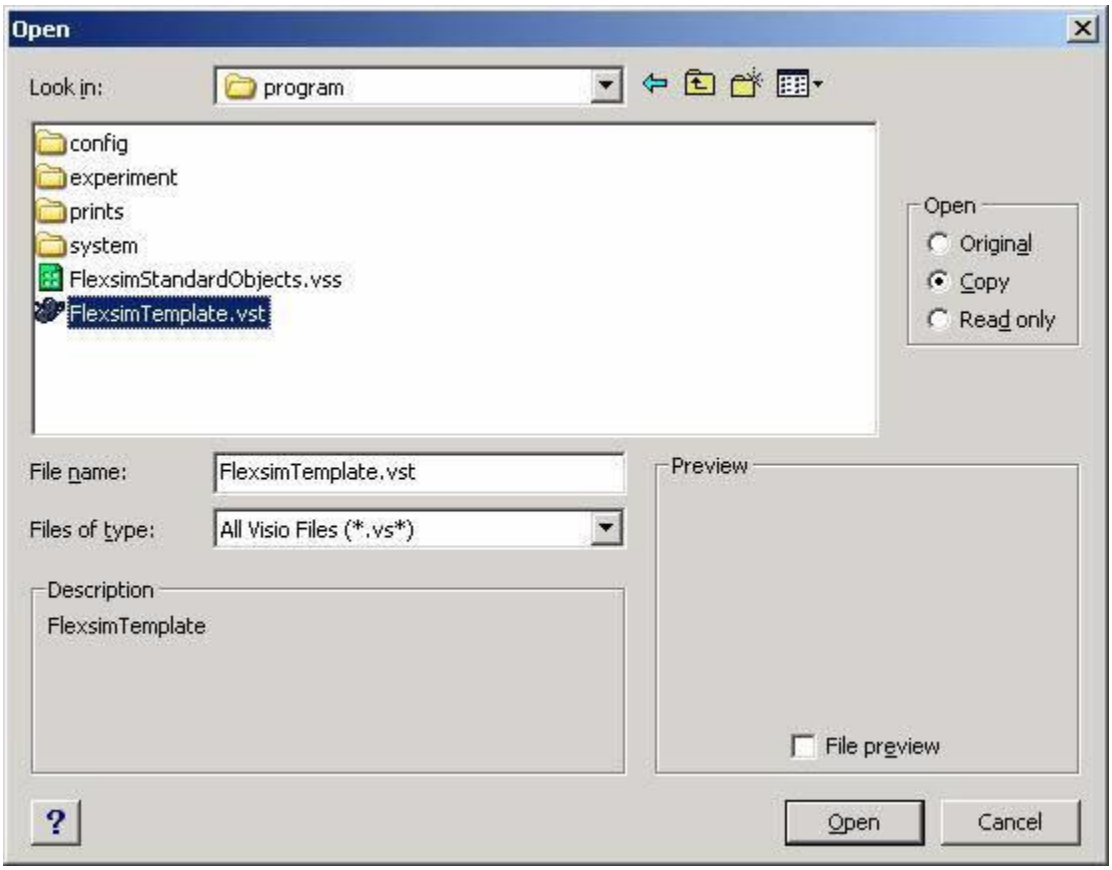
The Visio import tool allows you to build models in Microsoft Visio™ and import them into FlexSim.

Tutorial

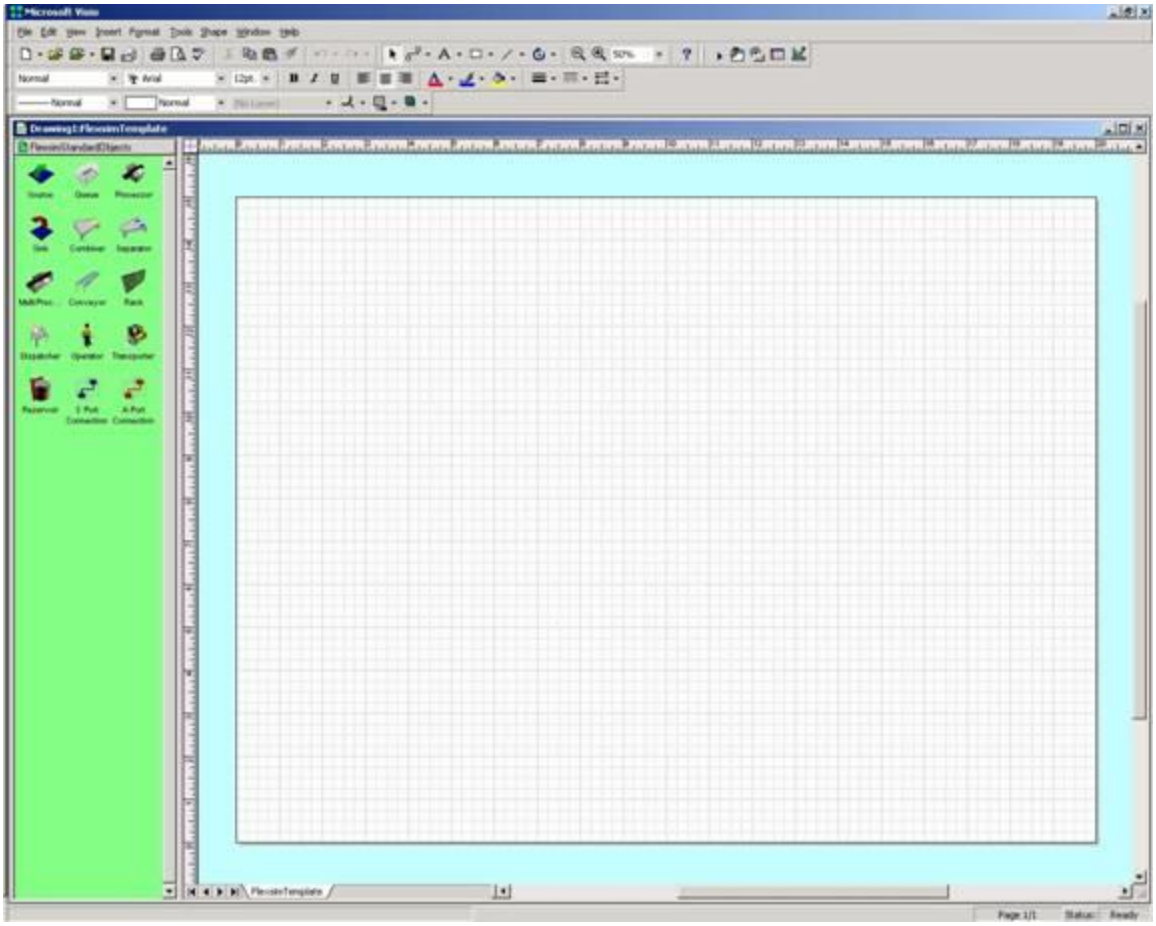
Start Visio



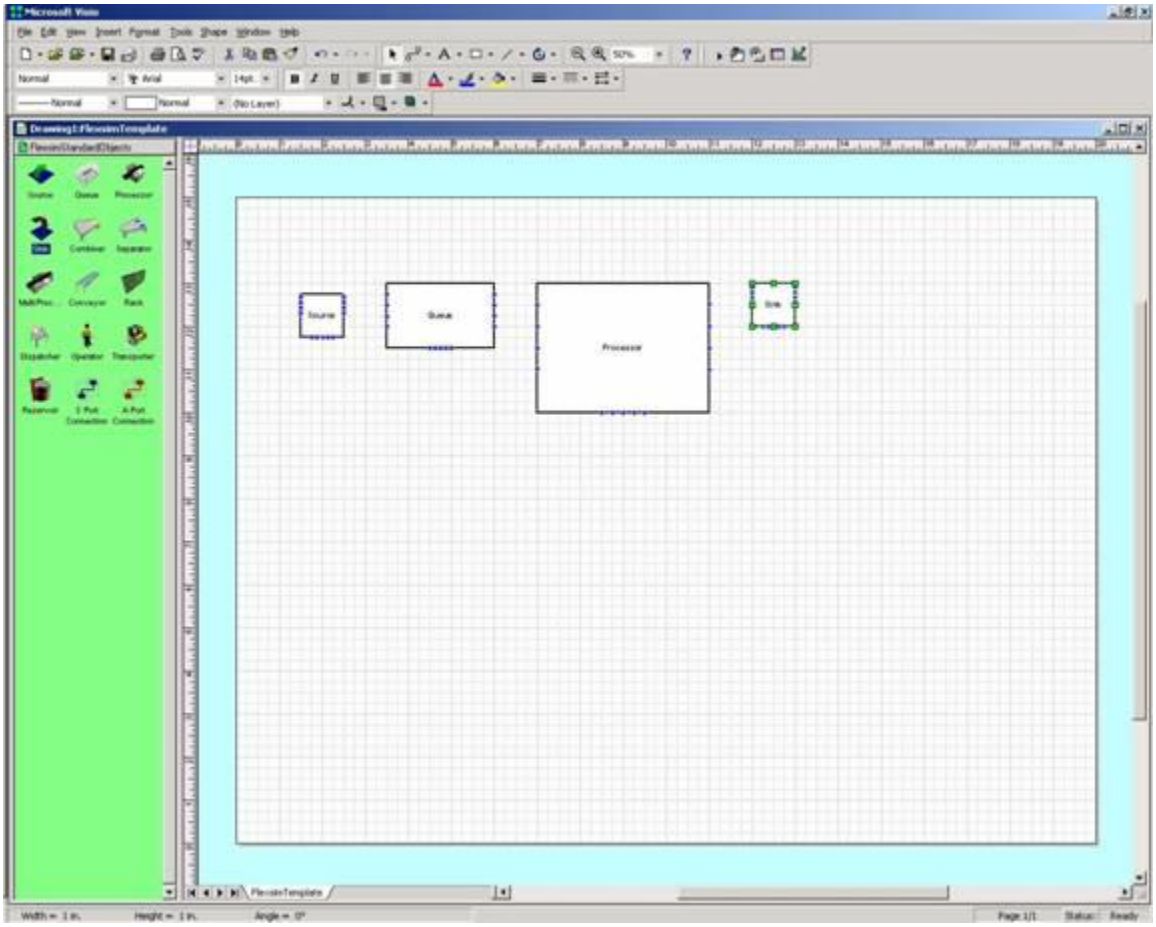
Open the FlexsimTemplate.vst



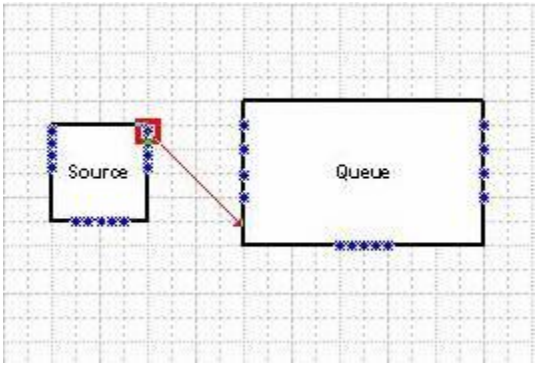
Visio will look like this.



Drag objects from the Stencil into the model space to arrange their layout.

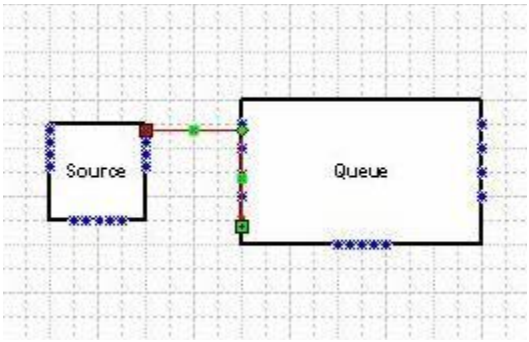


Drag the A Port Connection object into the model to create a connection.

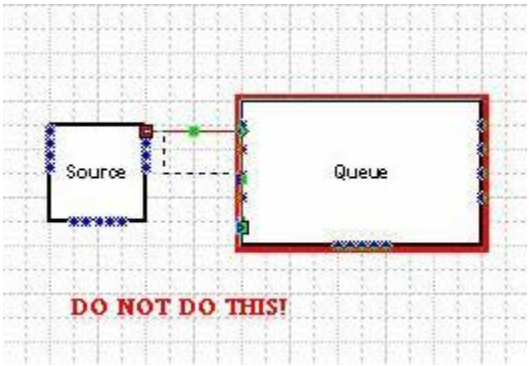


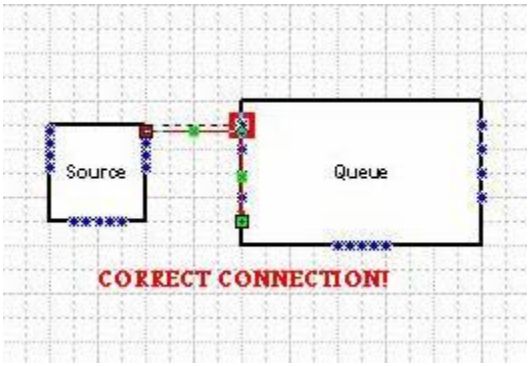
The connection is made from the upper left point to the bottom right point. If you hook the connection backwards, the connection will be backwards in FlexSim.

This is a connection that is only connected to the upstream object.



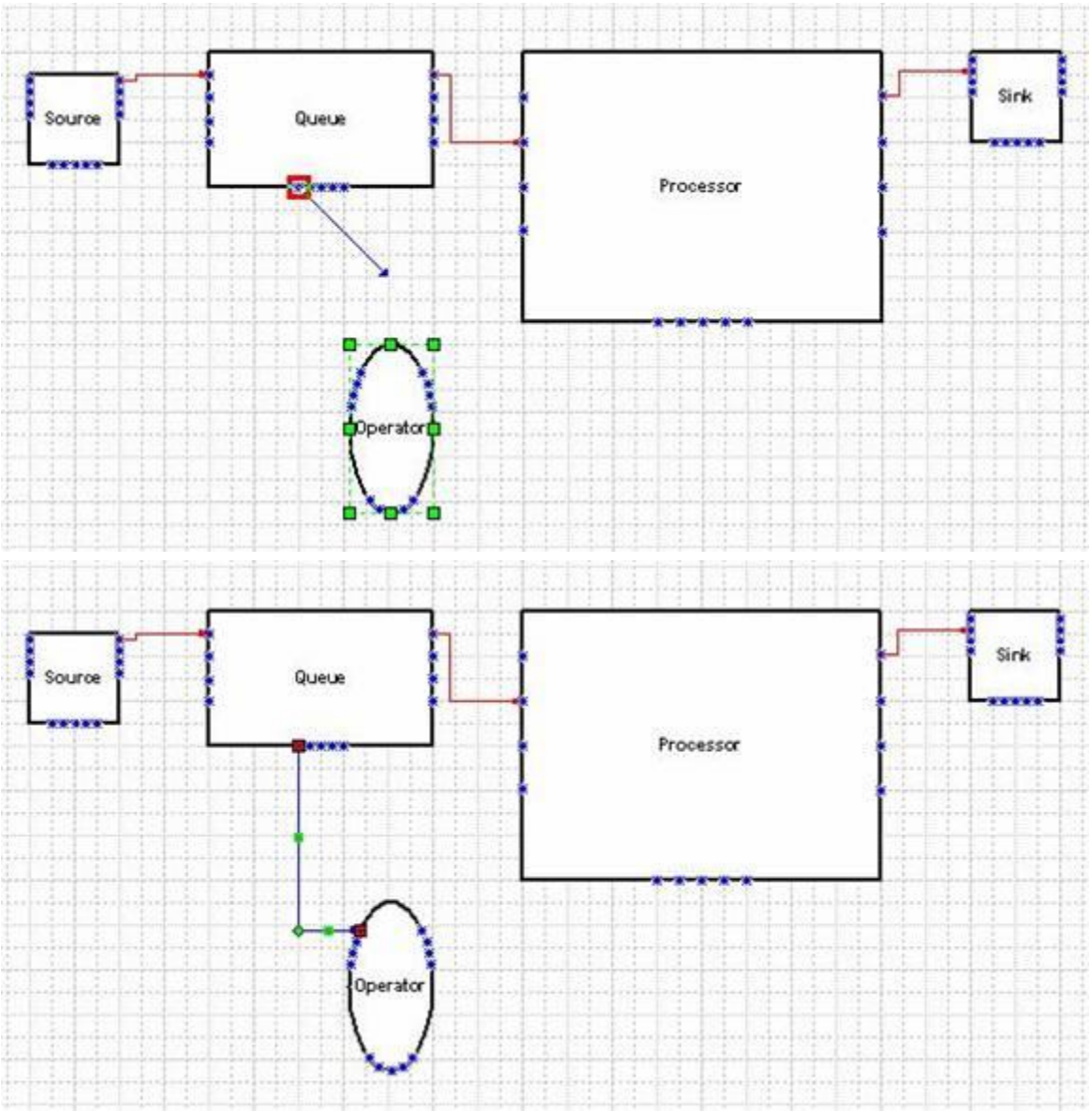
Note: Do NOT connect the connection directly to the object. It must be connected to one of the blue points.





When dragging out a connection, it is acceptable to have both ends touching blue connection points. The connection will be created correctly. Also, the connection point that you connect them to in Visio does not matter. FlexSim connections will be ordered according to the order in which they were dragged out in Visio. The example below will connect Queue to Processor's first input port despite being connected to the second connection point on the object in Visio.

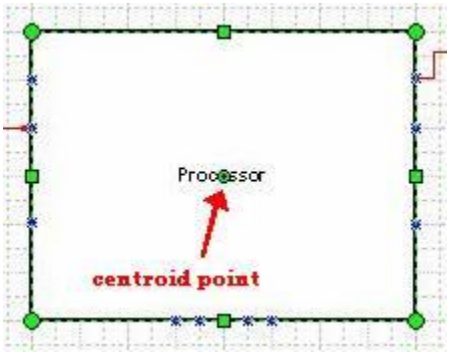
S port connections (center ports) are created in the same way as A port connections. The connection point in Visio that they are connected to does not make a difference in their order. Connecting S port connections to the blue connection points on the left and right is acceptable.



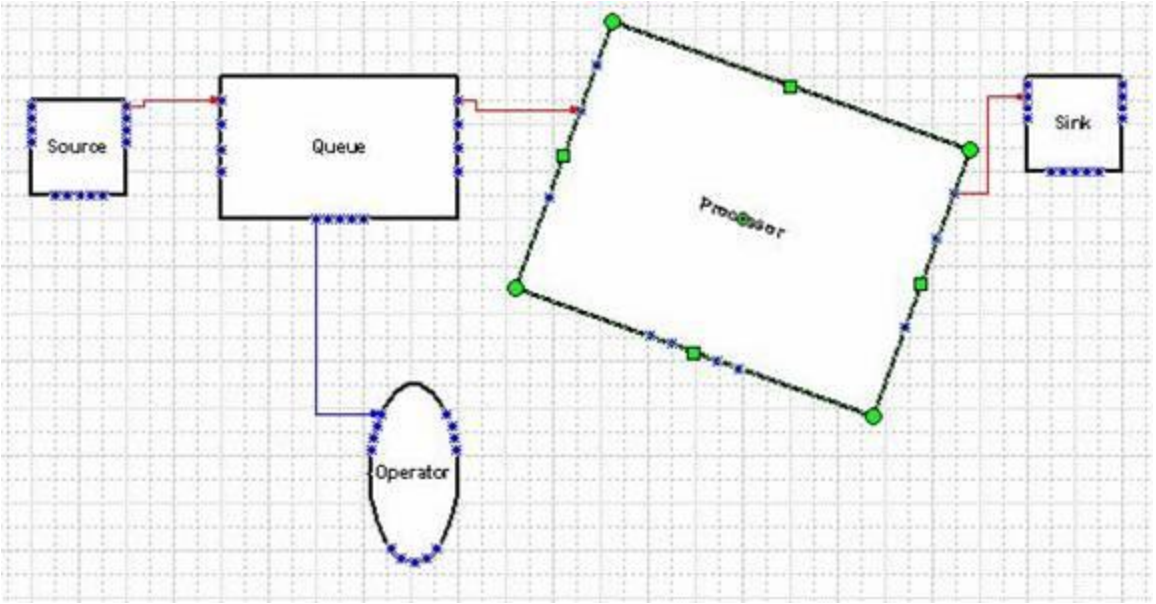
Objects can be rotated and resized from Visio to resize and rotate objects in FlexSim.

The rotation tool in Visio looks like this.

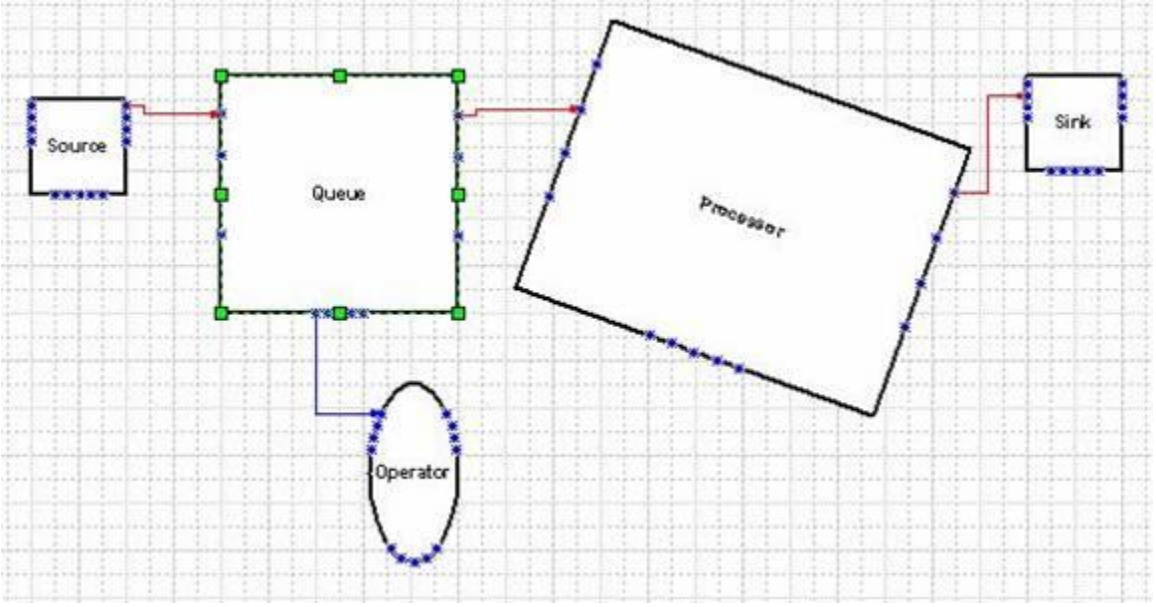
Do not move the centroid point of the object when resizing. The centroid point is the green circle with a black point in its middle. It is used to define where the center of rotation is for that object. If you move it in Visio, the object will not be located correctly in FlexSim.



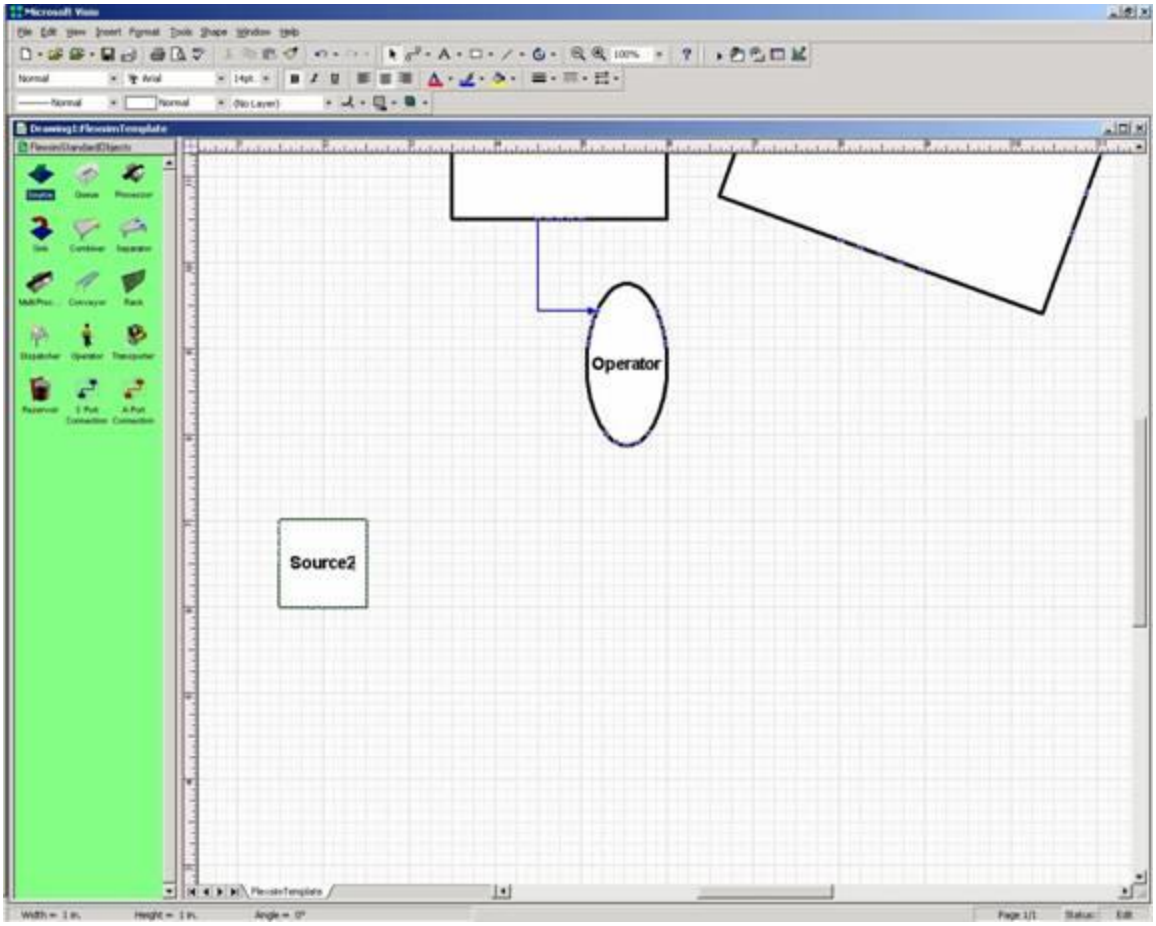
This processor was rotated by clicking and dragging the rotation tool around one of its corner points.



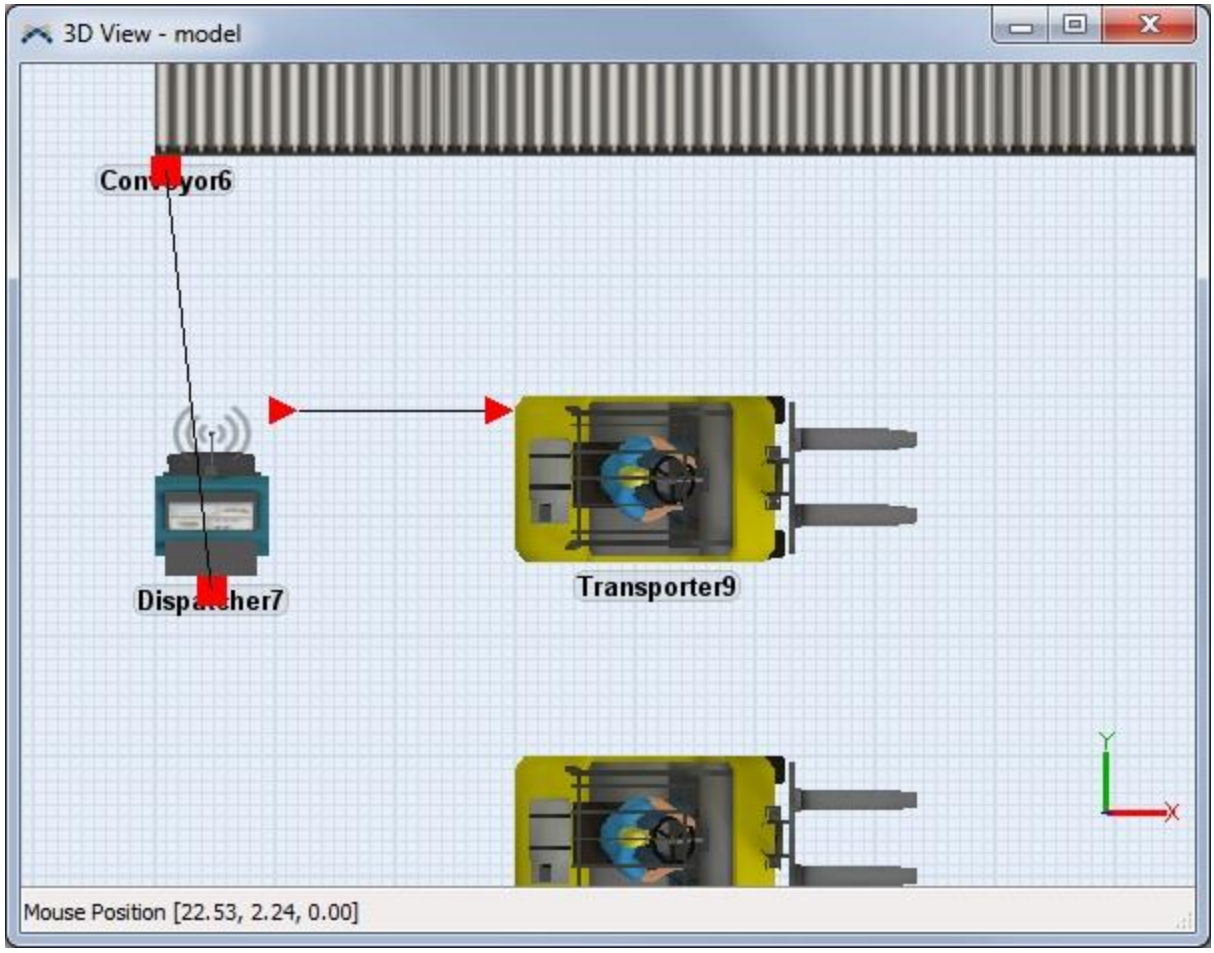
This queue has been resized by dragging its side point down using the selection tool .



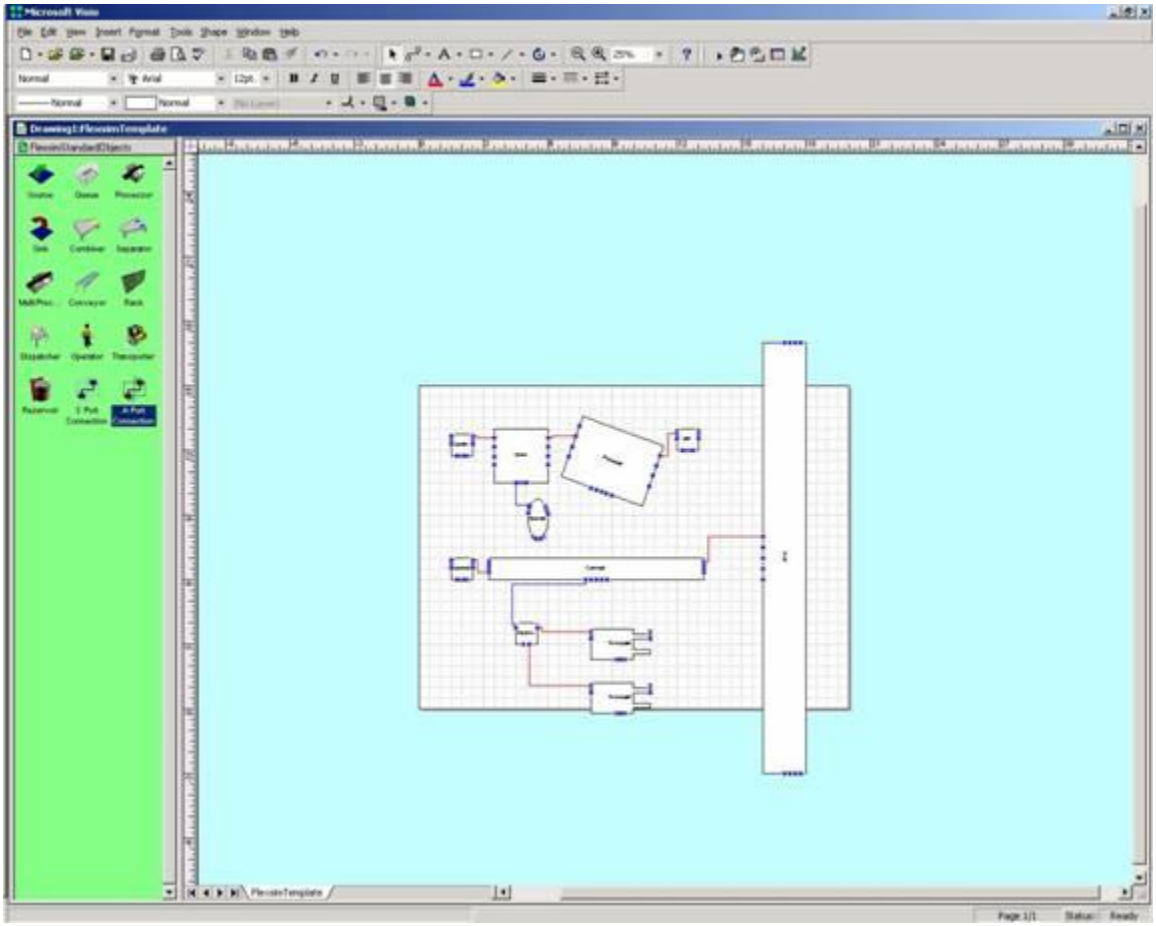
Objects can be renamed by double clicking on the object and typing in a new name.



Each object in your model must have a unique name in order for it to be correctly imported into FlexSim. If objects are given the same name in Visio, their connections will not be created correctly in FlexSim. In the example below, the Transporters have the same name, so the Dispatcher was connected to the first one twice.

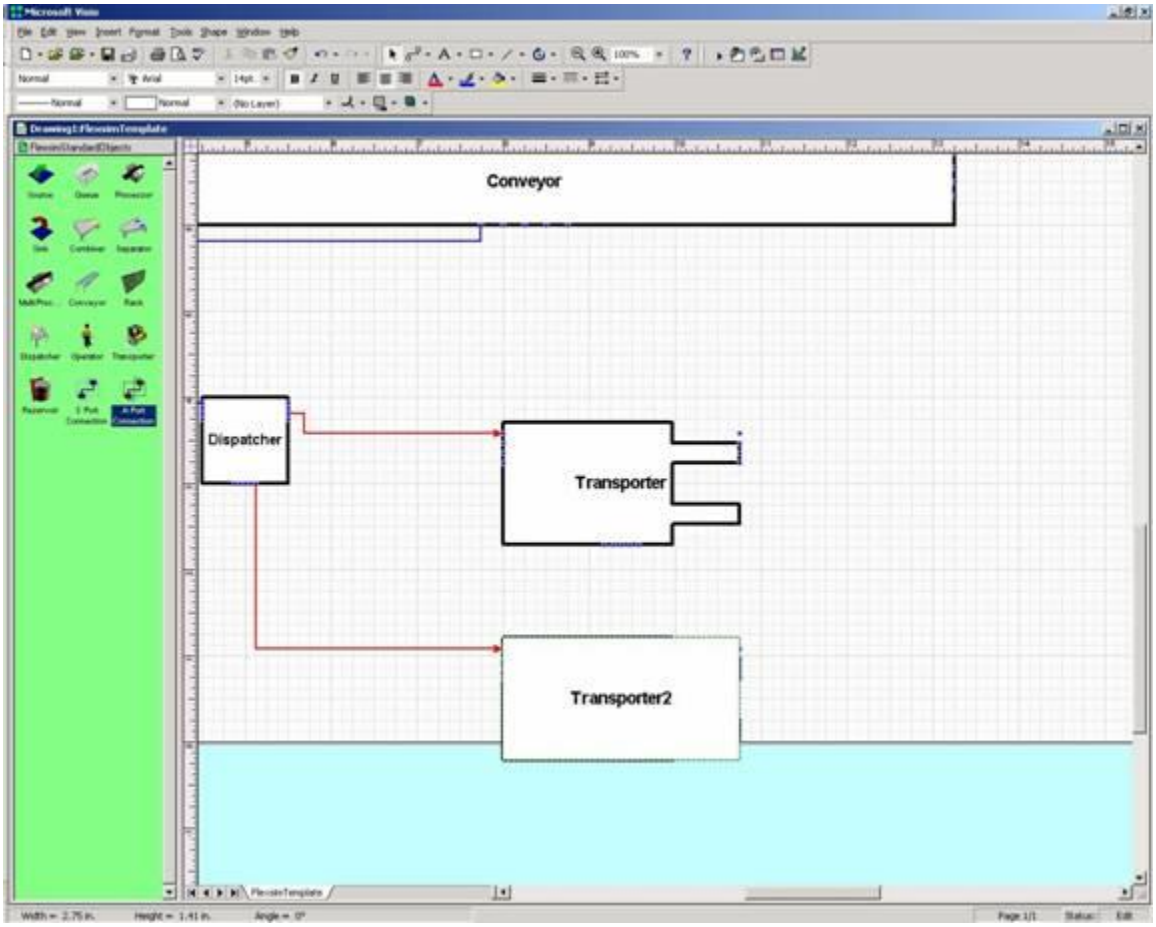


Here is a nearly completed Visio page with a FlexSim layout.

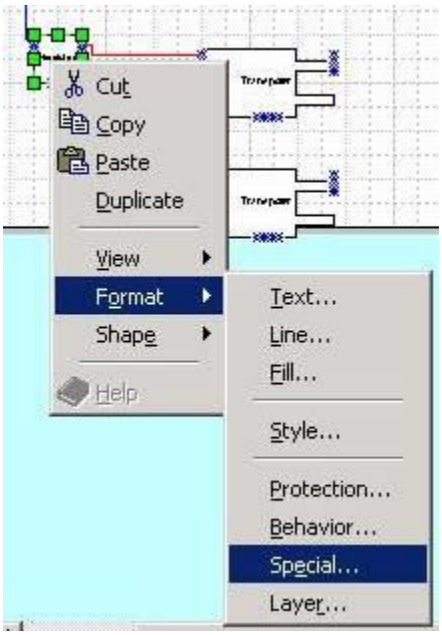


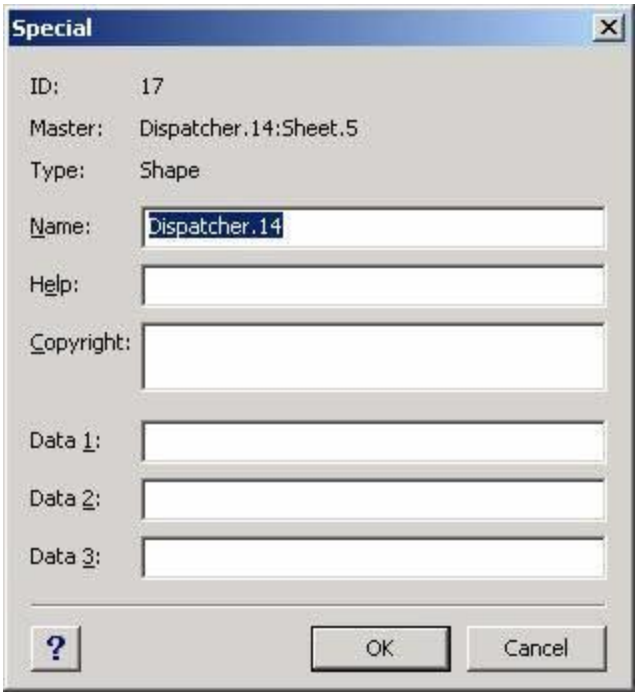
There are a few more things to check to be sure are correct before exporting it to Excel to import into FlexSim.

Be sure each object has a unique name.

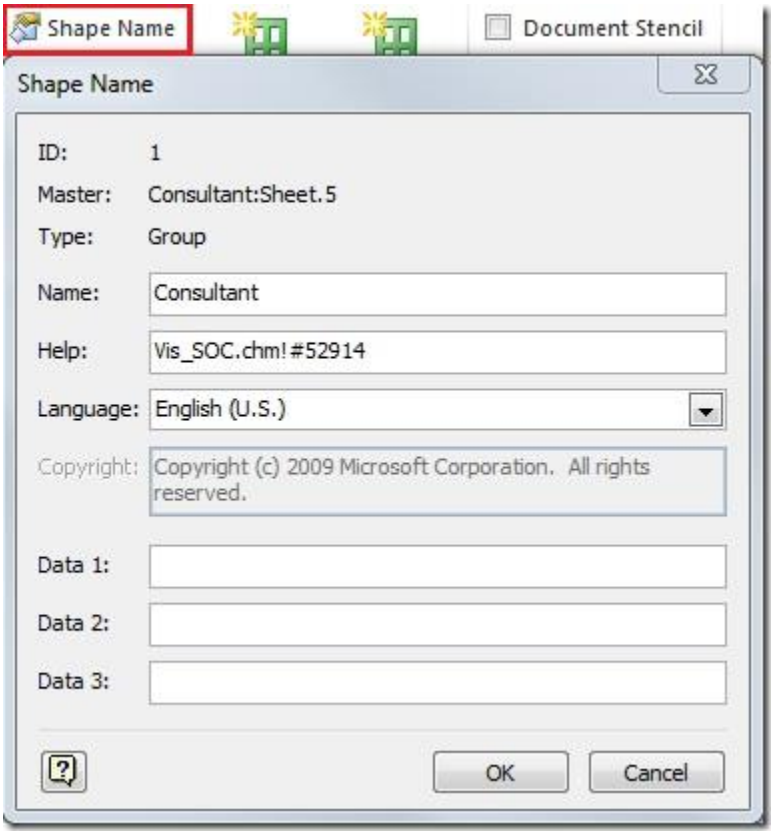


and each object must be correctly formatted in this menu.

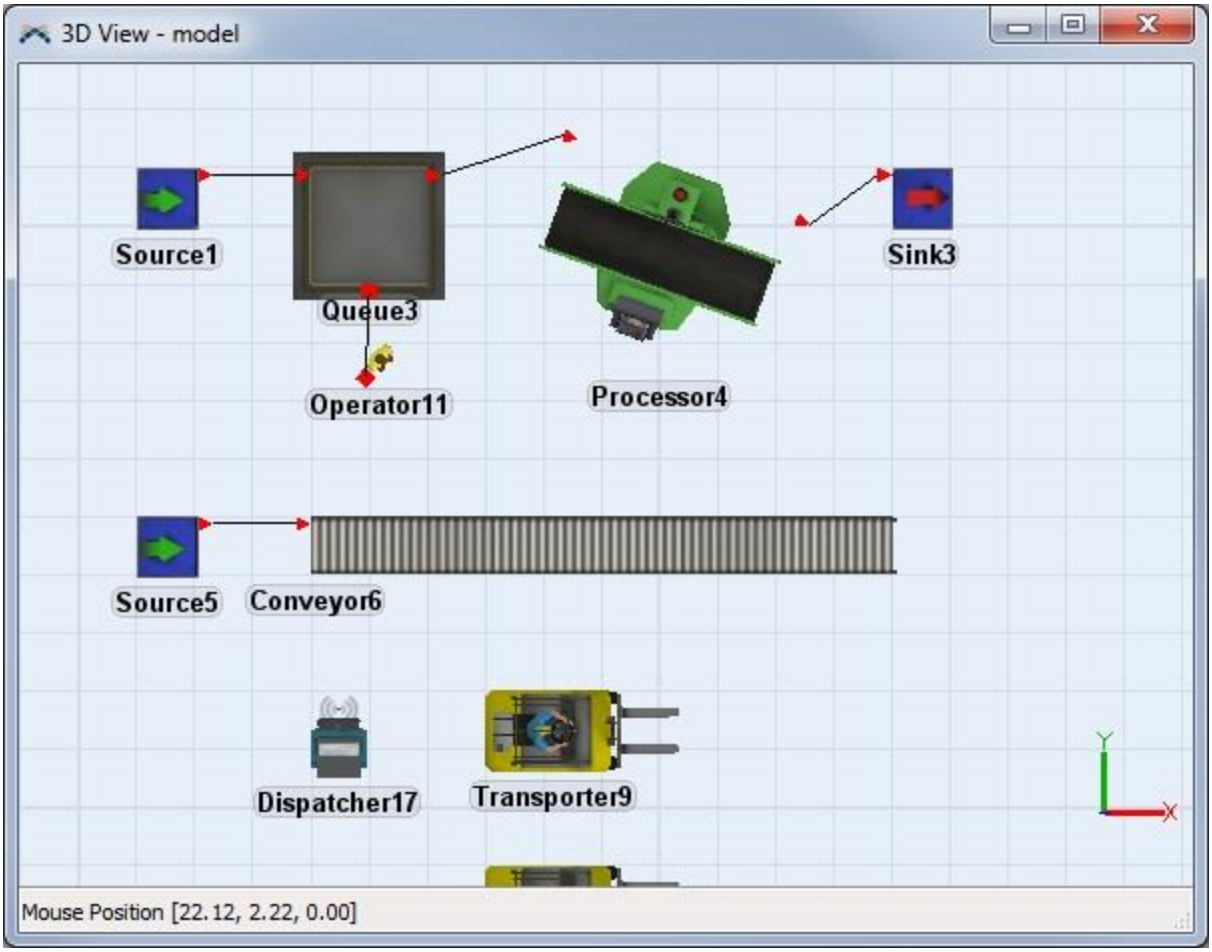




In Visio 2010 this dialog box has been renamed “Shape Name” as seen in the picture below

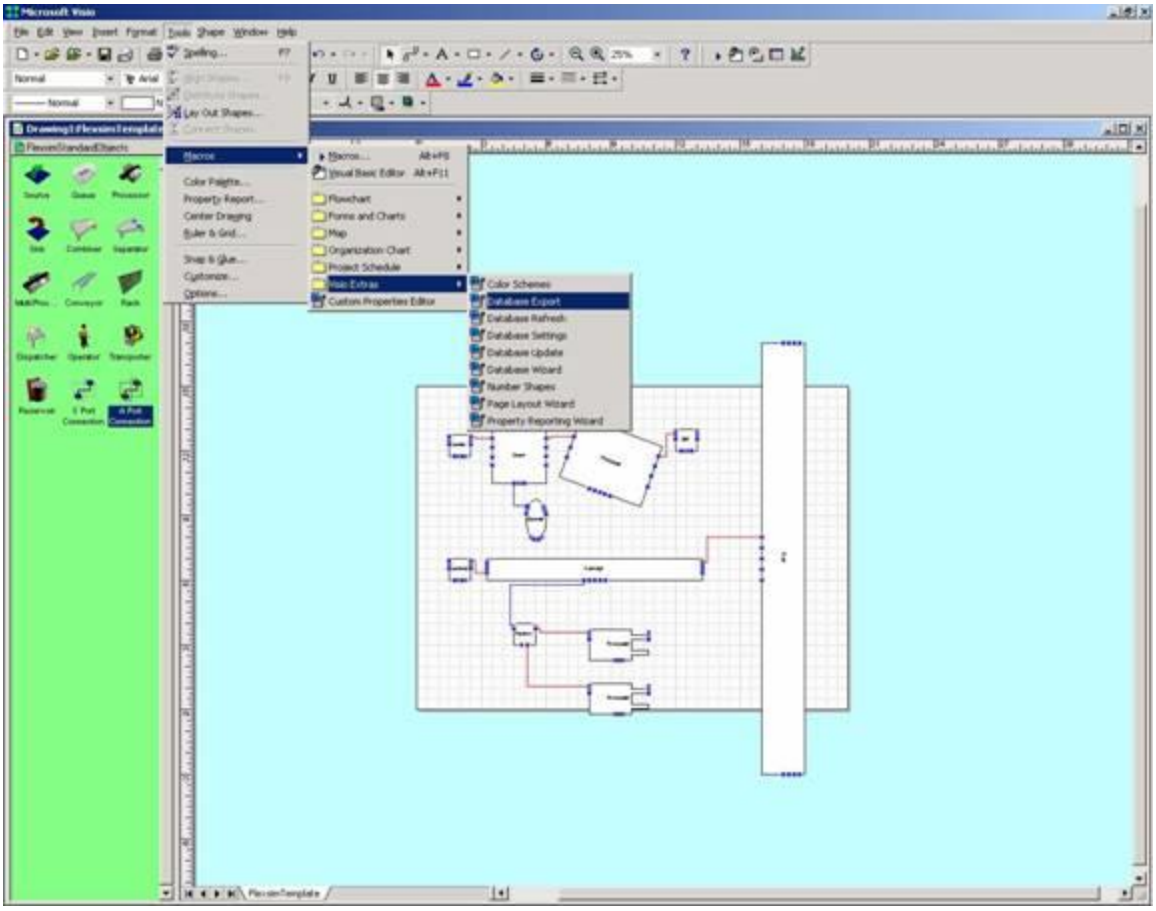


The Name field contains the object’s class and a number separated by a period or just the object’s class. If there is a number, it must match the ID number or the object will not be correctly exported. Typically the objects do this naturally, but sometimes Visio will incorrectly name an object in this field. If your model didn’t correctly import into FlexSim, this was probably the reason why. The example below shows what happens if the name is left as “Dispatcher.14” instead of changed to “Dispatcher.17”.

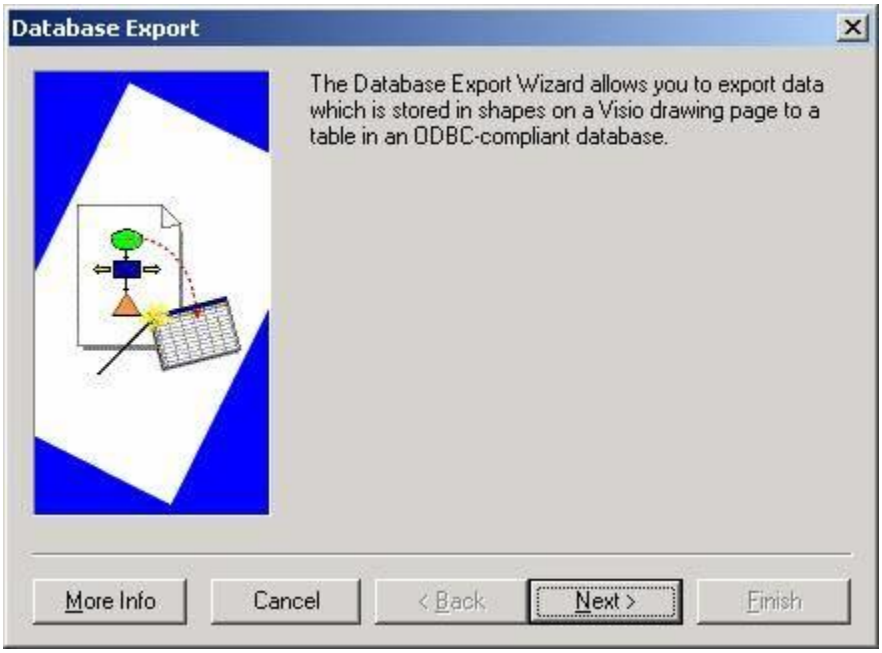


Now that everything has a unique name, and is correctly named in the Special Format, we are ready to export.

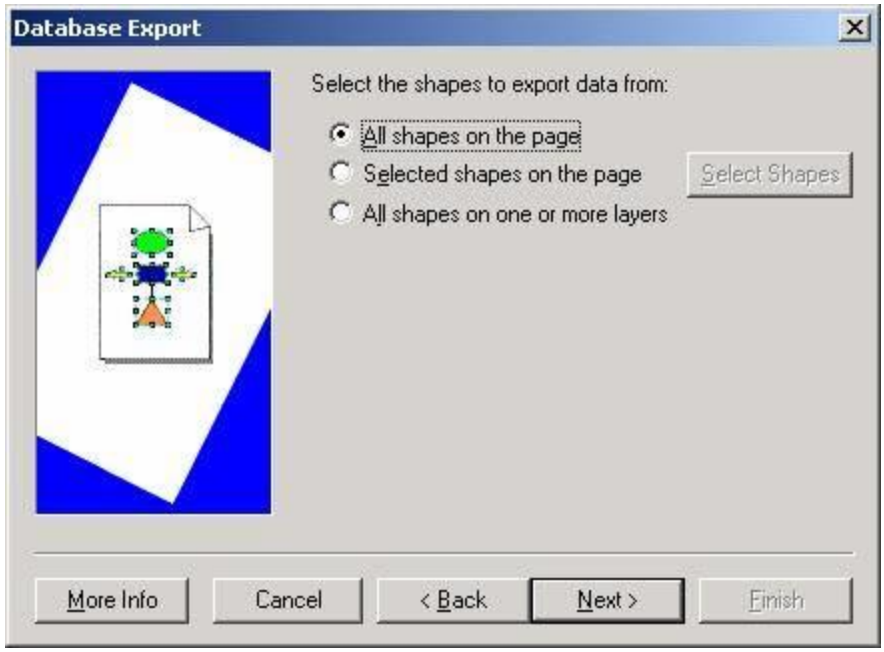
Go to the Tools > Macros > Visio Extras > Database Export to export the file.



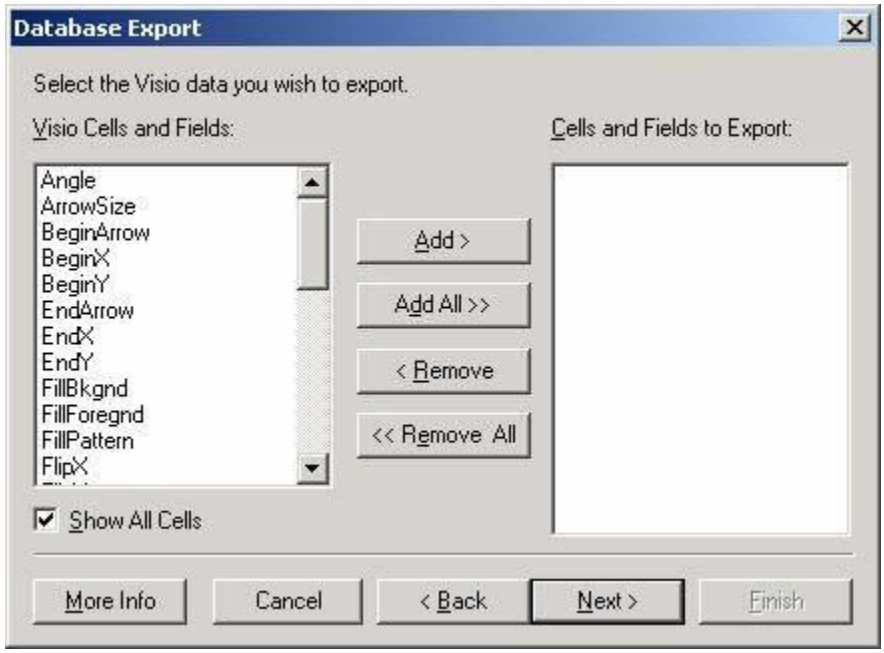
This screen will appear. Click next.


Database Export
The Database Export Wizard allows you to export data which is stored in shapes on a Visio drawing page to a table in an ODBC-compliant database.
More Info
Cancel
< Back
Next >
Finish

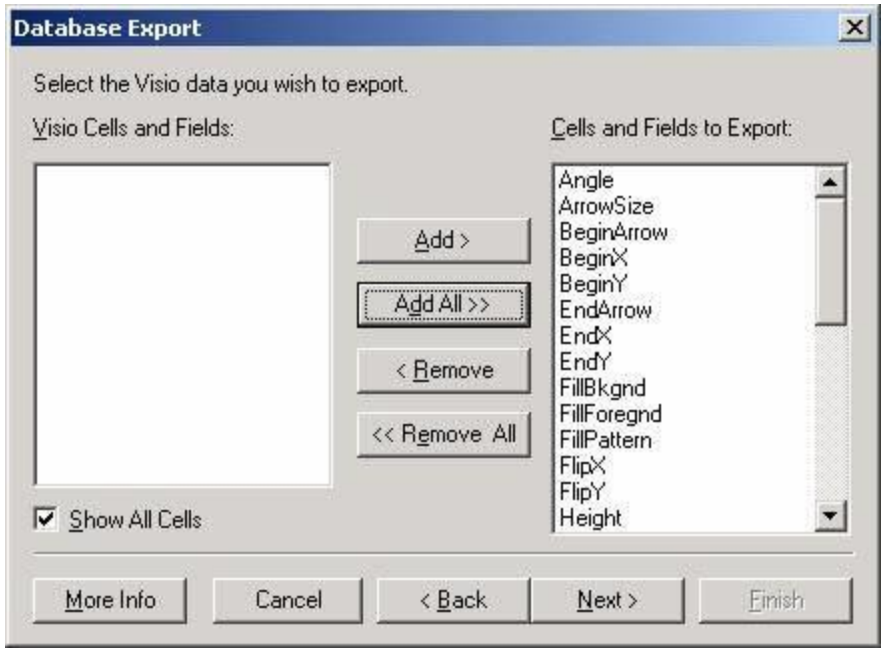
Select “All shapes on the page” and click Next.



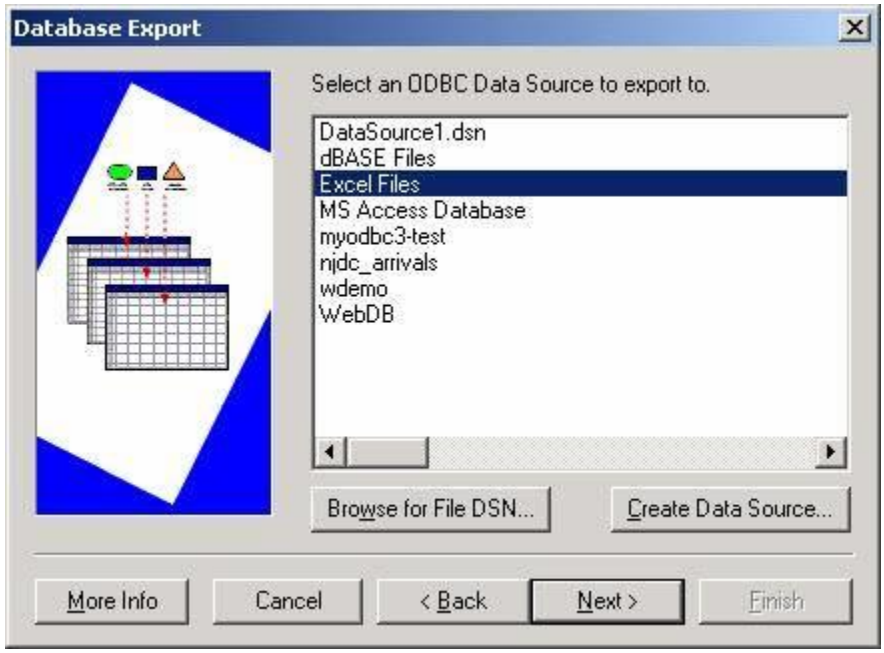
Click the “Add All >>” button to get all the correct information exported.



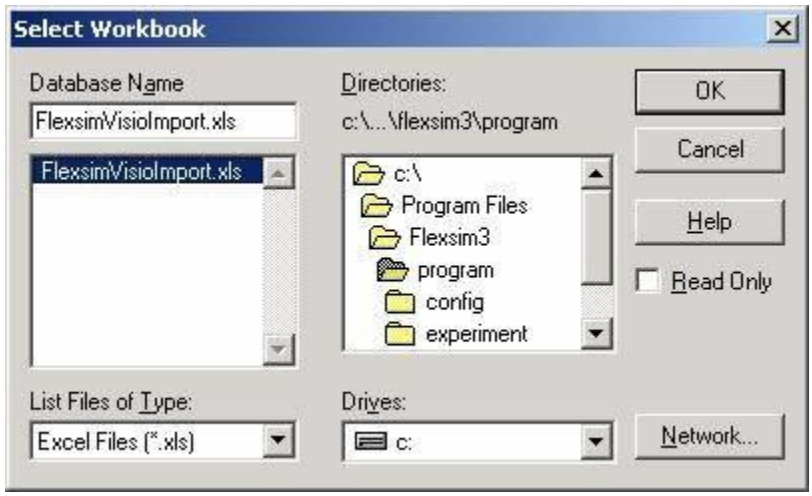
Then click Next.



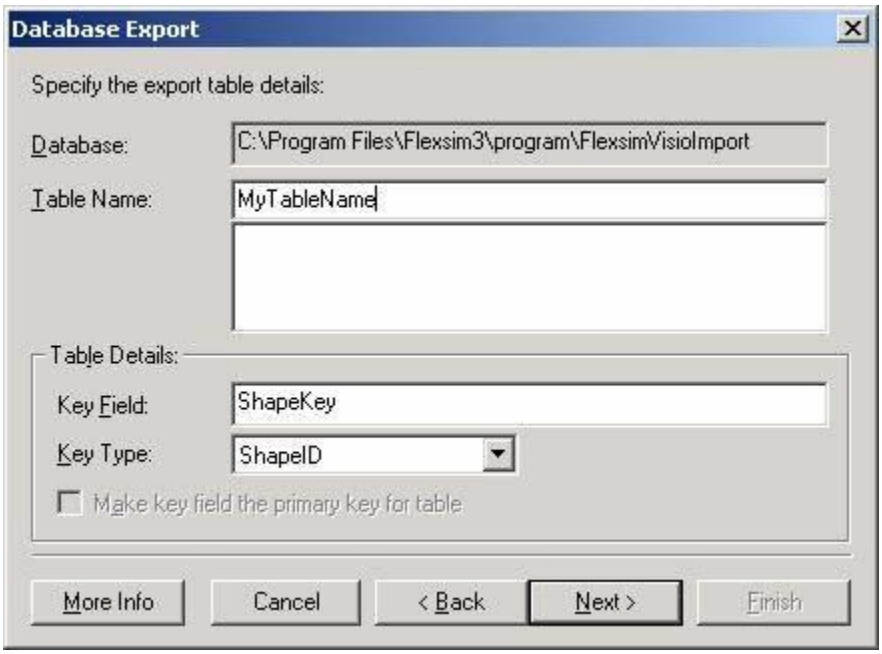
Select Excel Files and click Next.



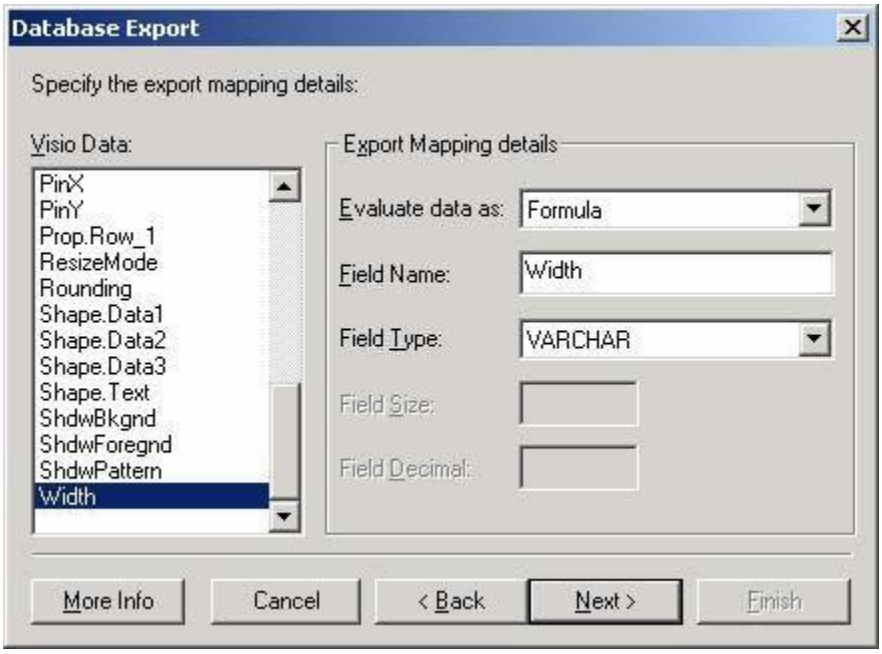
Select the excel file to export to. This should usually be C:/Program Files/Flexsim3/program/FlexsimVisioImport.xls.



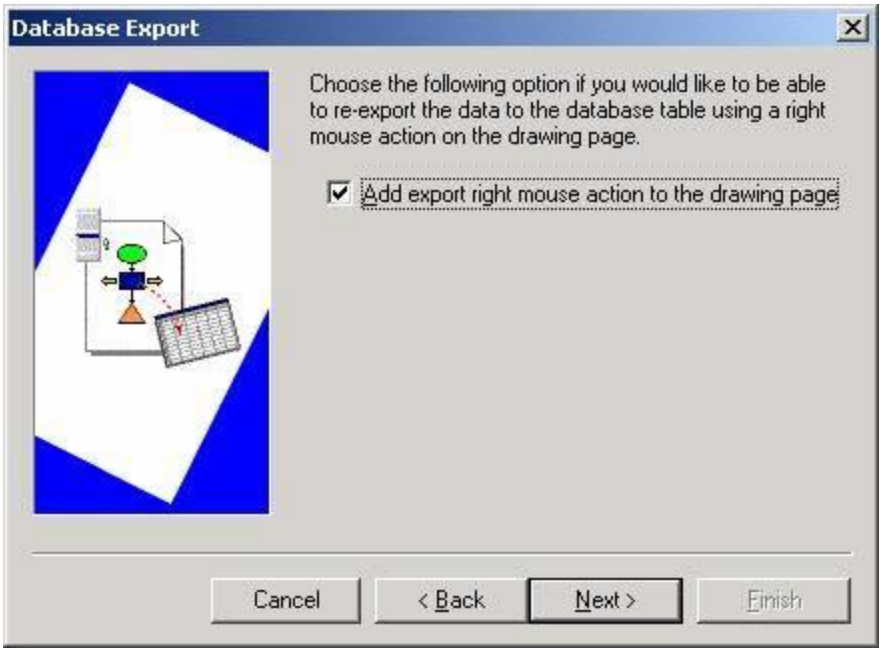
Type a name into Table Name. This name will be typed again into FlexSim so that it knows which sheet to import from. Remember the name you typed and click Next.



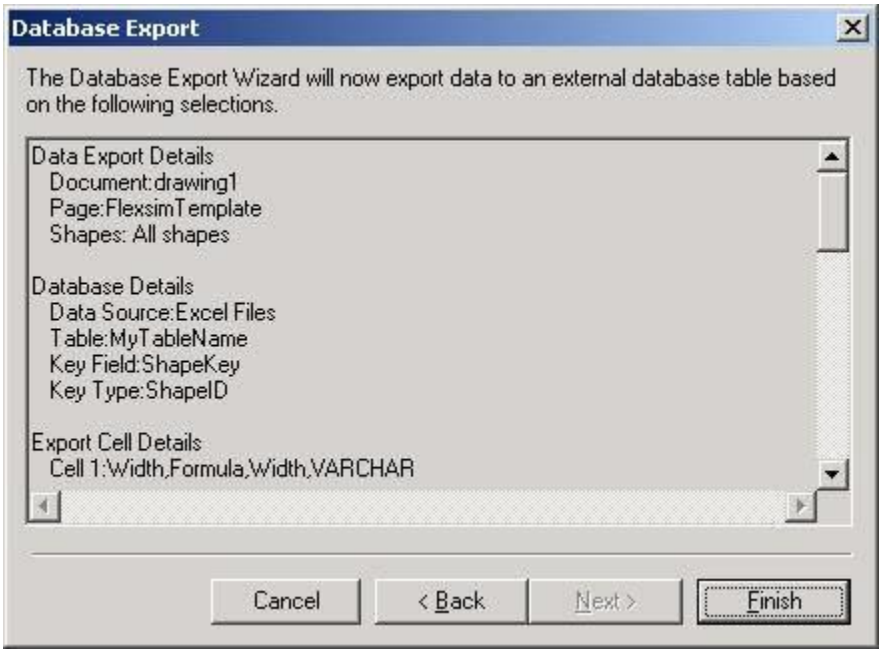
Do not change any values for the data on this screen. Click Next.



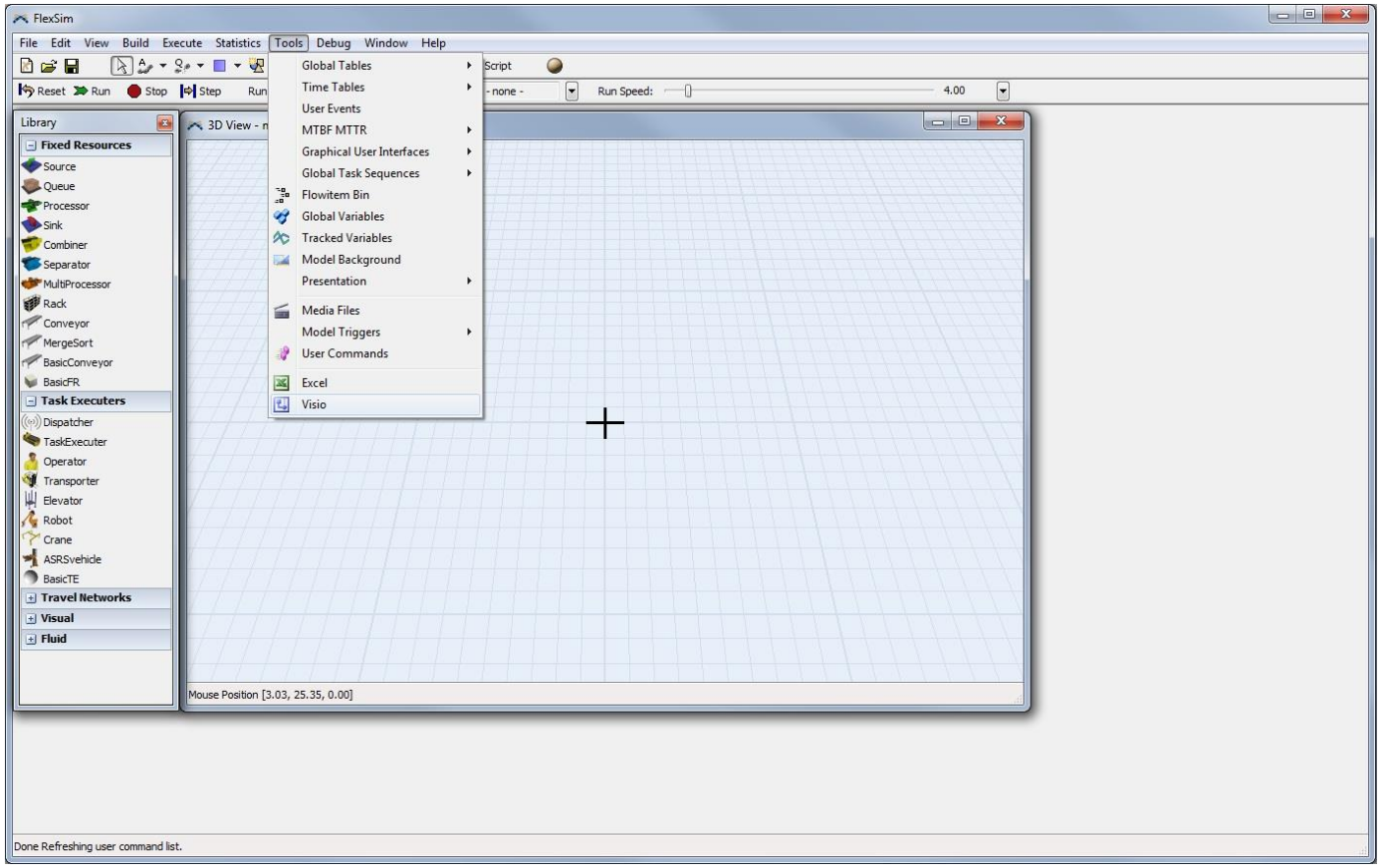
Next.


Database Export
Choose the following option if you would like to be able to re-export the data to the database table using a right mouse action on the drawing page.
Add export right mouse action to the drawing page
Cancel
< Back
Next >
Finish

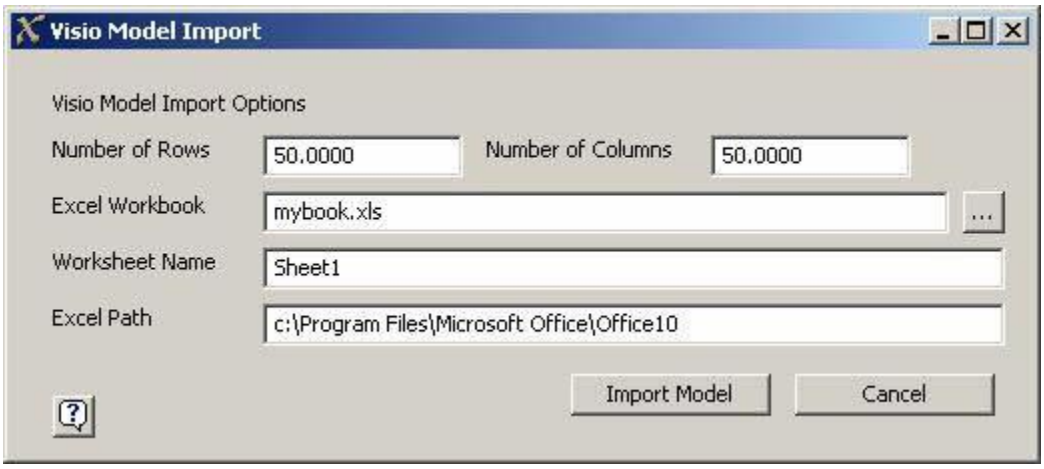
Finish.



Now, in FlexSim go to Tools > Visio…

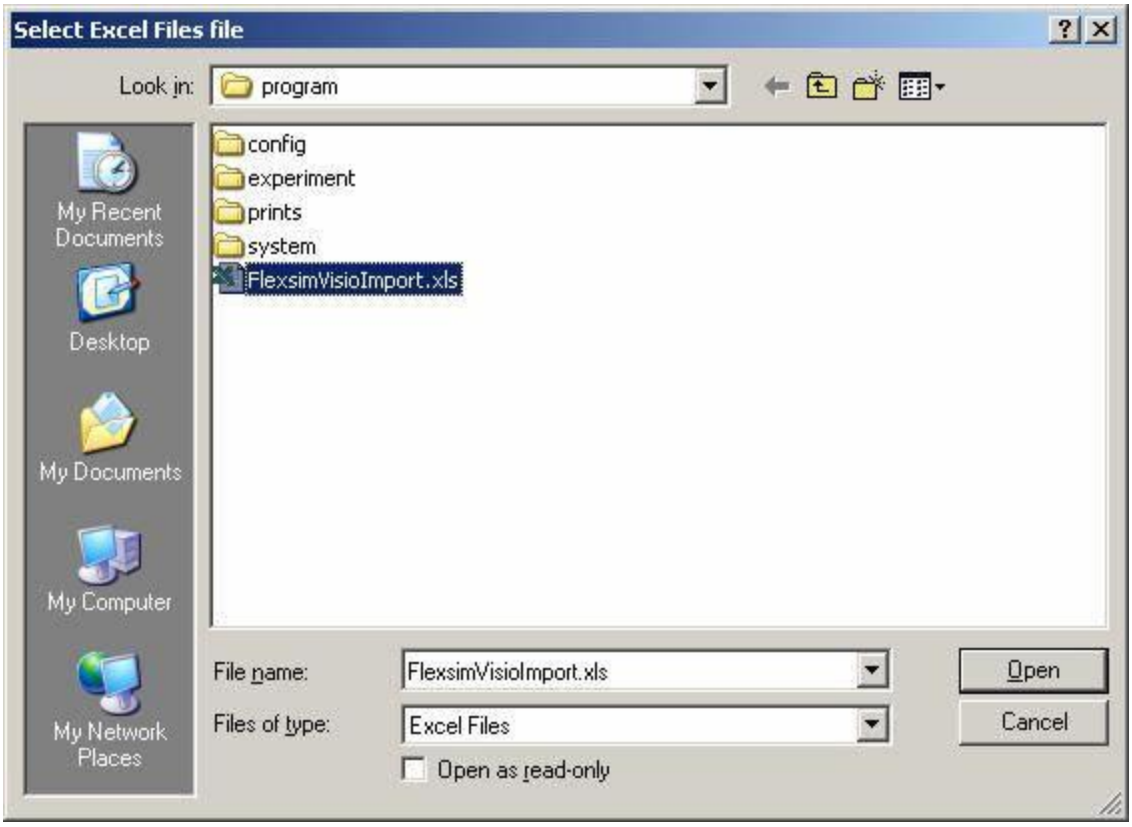


This dialog box will appear.

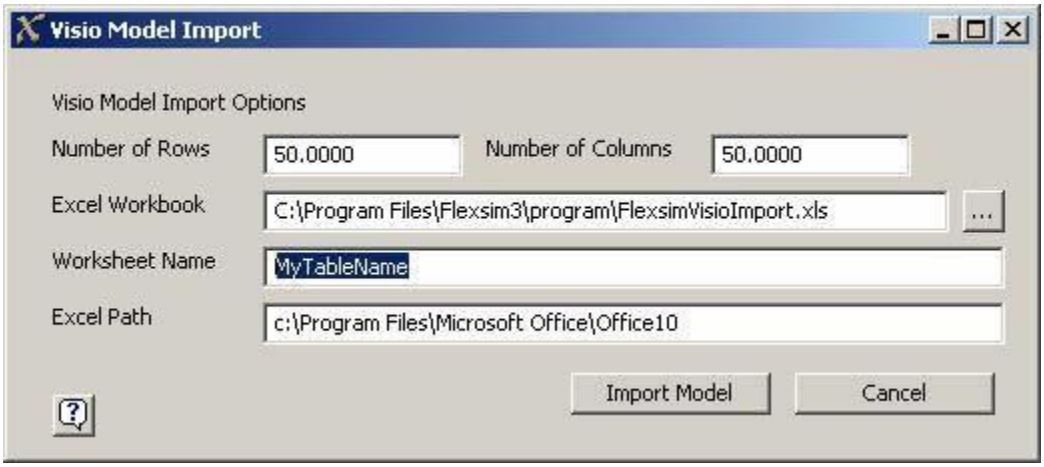


Enter a number into Number of Rows and Number of Columns that is larger than the number of rows and columns in the excel sheet that was just created by Visio Database Export. 50 is usually large enough. If your model does not import correctly, you may need to increase these values after checking your excel sheet to see how large it is.

Click the … button to browse for the correct excel file to open.

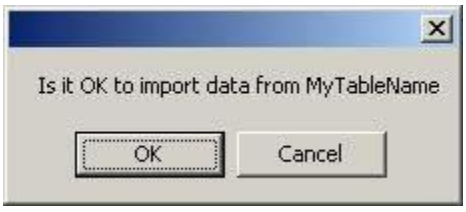


Type the Table Name that was entered earlier into the space for Worksheet Name.



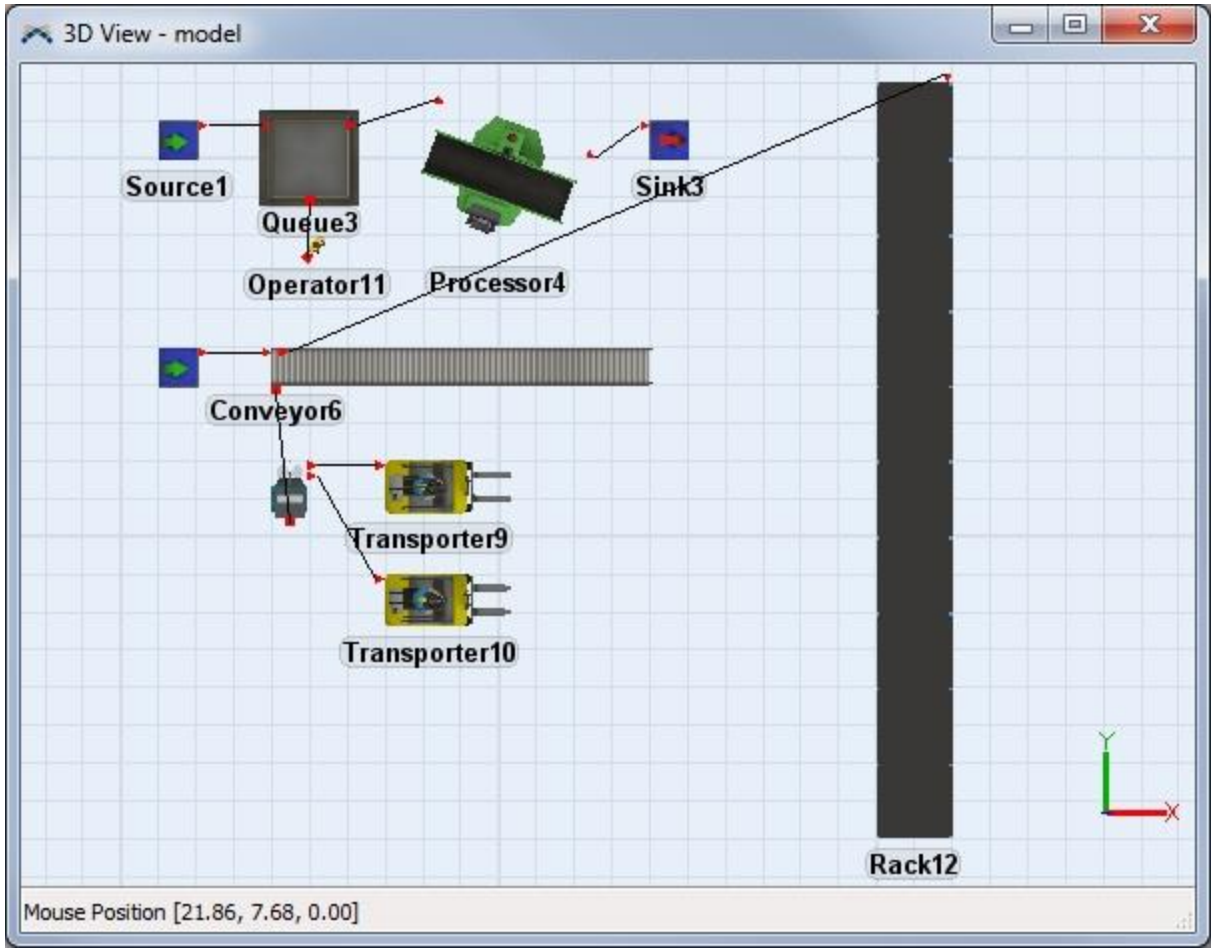
Be sure that the correct Excel Path is specified.

Click the Import button. This message box will appear.



Wait until Excel has completely opened the correct worksheet and then click OK.

The model will then be imported. This may take a few minutes. Wait for the Visio Model Import window to close. Your model will then be imported into FlexSim.


3D View - model
Source1
Queue3
Operator11
Processor4
Sink3
Conveyor6
Transporter9
Transporter10
Rack12
Mouse Position [21.86, 7.68, 0.00]

Pick Lists

1. Triggers
2. Time Pick Lists
3. Fixed Resources
4. Task Executers
5. Experimentation
6. Other

Triggers

1. Breakdown/Repair Trigger
2. Collision Trigger
3. Creation Trigger
4. Down/Up Trigger
5. Entry/Exit Trigger
6. Load/Unload Trigger
7. Message Trigger
8. Node Entry Trigger
9. OnChange Trigger
10. OnCover / OnUncover Trigger
11. OnDraw Trigger
12. OnEmpty/OnFull
13. OnEntryRequest
14. OnReceiveTaskSequence
15. OnResourceAvailable
16. Process Finish Trigger
17. Reset Trigger

Breakdown/Repair Trigger (On Break Down/On Repair)

Overview:

Breakdown Trigger: This code is executed each time this object goes down.

Repair Trigger: This code is executed each time this object finishes its repair time.

Access variables:

current: the current object

Collision Trigger (Handle Collision)

Overview:

The collision trigger is fired whenever an object executes its collision check and finds that it has collided with one of its collision members. For more information on collision detection, refer to TaskExecuter collision detection.

Access variables:

thisobject: the current object

otherobject: the object that the current object collided with

thissphere: the involved sphere of the current object

othersphere: the involved sphere of the object that the current object collided with

Creation Trigger (On Creation)

Overview:

This code is executed when a flowitem is first created.

Access variables:

current: the current object

item: the object that was just created

rownumber: the row number of the arrival (if one applies)

Down/Up Trigger (On Break Down/On Repair)

Overview:

Down Trigger: This function is executed when the MTBFMTTR object tells its members to break down.

Up Trigger: This function is executed when the MTBFMTTR object tells its members to come back up.

Access variables:

current: the MTBFMTTR object

members: the members list in the MTBFMTTR object

involved: either 1) the MTBFMTTR object or 2) the involved member

curmember: the current member (will cycle through all members) - not available in all options

index: the rank of curmember in the members list

Entry/Exit Trigger (On Entry/On Exit)

Overview:

Entry Trigger: This function is executed each time a flowitem enters this object.

Exit Trigger: This function is executed each time a flowitem exits this object.

Note on entry/exit trigger context: Generally, the entry and exit triggers are executed at the very beginning of the OnReceive/OnSend event of an object, before any other logic is executed. This means that you can change the object's variables, labels, etc., and have those changes be applied correctly within the event logic. However, executing commands that may affect further events of the object should not be executed in the entry/exit trigger, because some events have yet to be created in the event logic of the object, and functions which affect the object's events should wait until those events have been created. In such a case you should send the object a delayed message in 0 time (using the senddelayedmessage() command), and then execute the functionality from the message trigger. This allows the object to finish the rest of its event logic before your commands are executed. Commands which apply in this setting are stopobject(), requestoperators(), openoutput(), openinput(), resumeinput(), resumeoutput(), and in some cases the creating and dispatching of task sequences, depending on the types of tasks that are in them.

Access variables:

current: the current object

item: the involved object that just entered/exited

port: the number of the port that the object came/left through

Load/Unload Trigger (On Load/On Unload)

Overview:

Load Trigger: This trigger is fired once the TaskExecuter has finished its loadtime and just before it moves the flowitem into the TaskExecuter.

Unload Trigger: This trigger is fired once the TaskExecuter has finished its unloadtime and just before it moves the flowitem into its destination.

Access variables:

item: the item that is about to be loaded/unloaded

current: the current object

station: the station where the current object is loading from or unloading to

Message Trigger (On Message)

Overview:

This function is executed when a message is sent to this object by the sendmessage or senddelayedmessage commands. Each command may pass up to three user-defined parameters.

Access variables:

current: the current object

msgsendingobject: the object that sent the message

msgparam(1): the message's first value parameter

msgparam(2): the message's second value parameter

msgparam(3): the message's third value parameter

Node Entry Trigger (On Continue/On Arrival)

Overview:

This function is executed when a traveler enters this node from any direction. OnArrival happens when a traveler arrives at the node. OnContinue happens when the traveler continues down the next path.

Access variables:

current: the current object

traveler: the involved traveler that just entered

edgenum: the number of the edge that the traveler came through. The edge numbers for a given node are found by opening the node's properties, clicking on the General tab, then selecting Output Ports in the Ports section.

OnChange Trigger

Overview:

This function gets executed whenever any of the variables in the watchlist change. The station checks the variables on EVERY event in the model, so it is sure to catch a change when it happens.

Access variables:

current: the current object

changedobject: the object whose variable changed

changeditem: the variable (node) that was changed

changedvalue: the new value of the variable

oldval: the old value of the variable

OnCover OnUncover Trigger

Overview:

This function gets executed when the state of a photo eye changes.

Access variables:

current: the current object

item: the flowitem that is currently in front of the photo eye.

photoeye: the photo eye number. The photo eye numbers are found by opening the conveyors's properties, clicking on the Photo Eyes tab, then looking in the table within the Photo Eye Editor.

covermode: For a cover trigger, 1 means a green to yellow state transfer and 2 means yellow to red. For an uncover trigger, 1 means a yellow to green transfer and 2 means a red to green transfer.

OnDraw Trigger (Custom Draw Code)

Overview:

This function is executed prior to the object's OnDraw Event. It is used to perform user-defined drawing commands and animation. If this function returns 1, the object's standard OnDraw function is not called. If it returns 0, OnDraw occurs normally. FlexScript, C++, and/or OpenGL code can be used to define what is drawn.

Common Commands:

drawcolumn(xloc,yloc,zloc,nrsides,baseradius,topradius,height,xrot,yrot,zrot,red,green,blue[,opacity,texture,xrep,yrep])

drawcube(xloc,yloc,zloc,xsize,ysize,zsize,xrot,yrot,zrot,red,green,blue[,opacity,texture,xrep,yrep])

drawcylinder(xloc,yloc,zloc,baseradius,topradius,height,xrot,yrot,zrot,red,green,blue[,opacity,texture])

drawdisk(xloc,yloc,zloc,innerradius,outerradius,startangle,sweepangle,xrot,yrot,zrot,red,green,blue[,opacity,texture])

drawline(view,x1,y1,z1,x2,y2,z2,red,green,blue)

drawobject(view,shape,texture)

drawrectangle(xloc,yloc,zloc,length,width,xrot,yrot,zrot,red,green,blue[,opacity,texture,xrep,yrep])

drawsphere(xloc,yloc,zloc,radius,red,green,blue[,opacity,texture])

drawtext(view,text,xloc,yloc,zloc,width,height,thickness[,xrot,yrot,zrot,red,green,blue,opacity])

drawtomodelscale(object)

drawtoobjectscale(object)

drawtriangle(view,x1,y1,z1,x2,y2,z2,x3,y3,z3,red,green,blue)

spacerotate(x,y,z)

spacescale(x,y,z)

spacetranslate(x,y,z)

Access variables:

current: the current object

view: the view that the object is being drawn in

OnEmpty / OnFull Trigger

Overview:

These triggers are called on fluid objects when their content reaches 0 or when it reaches their maximum content. These triggers are often used to open or close ports or send messages to other objects in the model.

Access variables:

current: the current object

OnEntryRequest Trigger

Overview

This trigger is called by the TrafficControl object when another object makes an entry request.

Access variables:

current: the current object

traveler: the object which requested entry

OnReceiveTaskSequence

Overview

This trigger is called on any dispatcher object (including all task executers). It is fired whenever the object receives a task sequence. This trigger can be used to override the default passing logic, allowing more control over which object will receive the task sequence.

Access variables:

current: the current object

ts: the task sequence

OnResourceAvailable Trigger

Overview:

This trigger is available for either a Dispatcher or a TaskExecuter and behaves a little differently on each. For a Dispatcher, the trigger is fired whenever a downstream TaskExecuter becomes available. For a TaskExecuter, the trigger is fired whenever that TaskExecuter finishes a task sequence. If the TaskExecuter is also a Dispatcher, meaning it has a team that it dispatches tasks to, then the trigger will be fired for both cases.

If the function returns a 0, the Dispatcher/TaskExecuter will do its own dispatching logic. If the function returns a 1, the Dispatcher/TaskExecuter will not do anything, and assumes all dispatching logic is done with this trigger using the movetasksequence() and dispatchtasksequence() commands.

Access variables:

current: the current object

port: for a Dispatcher, this is the output port of the Dispatcher; for a TaskExecuter that has just finished a task sequence the port value is 0

resource: for a Dispatcher, this is the downstream resource that has become available; for a TaskExecuter, this value is the TaskExecuter itself (or the same as current)

nextts: this value is the next task sequence in the task sequence queue

lastts: for a Dispatcher, this value is always NULL; for a TaskExecuter, this value is the task sequence that was just completed before it became available

Setup/Process Finish Trigger (On Setup Finish/On Process Finish)

Overview:

This code is executed each time a flowitem finishes its setup/process time.

Access variables:

current: the current object

item: the object that has finished its setup/process time

Reset Trigger (On Reset)

Overview:

This function is executed when the model is reset.

Access variables:

current: the current object

Time Pick Lists

Note on distribution picklist options: Many of the pick options in these pick lists are distributions, and are not documented here. For more information on the different FlexSim distributions, refer to the ExpertFit documentation or the Commands Reference.

1. **Load/Unload Time**
2. **Minimum Staytime**
3. **Process Time**
4. **Setup Time**
5. **Time Picklist (Inter-Arrivaltime)**

Load/Unload Time

Overview:

Load Time: Returns the value of the load time. The flowitem is moved into the TaskExecuter at the end of the load time.

Unload Time: Returns the value of the unload time. The flowitem is moved into the station at the end of the unload time.

Access variables:

current: the current object

item: the involved flowitem

station: the object where the flowitem is being loaded or unloaded

Minimum Staytime (Minimum Dwell Time)

Overview:

This function returns the minimum time that each flowitem must reside in the Rack. When the flowitem enters the Rack, it executes this function, and creates an event in the returned. After the time has expired, the Rack releases the item. If this function returns a value of -1, the Rack will not create an event to release the item at all, and you will need to release the item explicitly using the releaseitem() command. This can be used if you want to implement your own releasing strategy for the Rack.

Access variables:

current: the current object

item: the involved flowitem

port: the port the product entered through

Process Time

Overview:

This function returns the process time for the processing object.

Access variables:

current: the current object

item: the involved flowitem

Setup Time

Overview:

This function returns the setup time for the processing object.

Access variables:

current: the current object

item: the involved flowitem

port: the port the product came in through

Time Picklist (Inter-Arrivaltime Usage)

Overview:

Returns the number of seconds between creation of flowitems.

Access variables:

current: the current object

FixedResources Picklists

1. Flow Rate
2. Item Speed
3. On Clear
4. On Cover
5. Pick Operator
6. Place in Bay
7. Place in Level
8. Pull Requirement
9. Pull Strategy
10. Rise/Fall Through Mark
11. Send Requirement
12. Send To Port
13. Split Quantity
14. Transport Dispatcher

Flow Rate

Overview:

Incoming Flow Rate: Returns the number of seconds between flowitems entering a reservoir. This function is executed whenever a flowitem enters. Here the volume of the entering flowitem should be determined. If the reservoir has been full and a flowitem exits and makes space available in the Reservoir, the in flow rate function is executed again from the OnSend event to determine the next time for the flowitem.

Outgoing Flow Rate: Returns the number of seconds between flowitems leaving a reservoir. Usually this function is executed when a flowitem exits, to find the time to release the next flowitem. However, the function is also called for the first flowitem that enters, to get the time to release that flowitem.

Access variables:

current: the current object

isentry: 1 means this function was executed from a flowitem's entry, 0 means it was executed from a flowitem's exit.

item: If isentry is 1, then this is a reference to the flowitem that just entered. Otherwise, it refers to the flowitem that just exited.

unitsnode: This is a reference to a node whose number value can be set to represent a volume for the item. Setting this node's value only applies if isentry is 1 and it is an incoming flow rate trigger, not an outgoing flow rate trigger.

Item Speed (Speed)

Overview:

This field is evaluated once for each flowitem when the flowitem enters. It returns the speed of that flowitem

Access variables:

current: the current object

item: the item that is ready to be sent

The item speed picklist uses the same options as the Time picklist, except the return value is interpreted as a speed instead of a time.

On Clear

Overview:

This field is evaluated once for each flowitem as the flowitem clears the decision point.

Access variables:

current: the current object

item: the item that is clearing the decision point

decisionpoint: the number of the decision point that caused this trigger to fire

position: the position of the decision point on the conveyor

On Cover

Overview:

This field is evaluated once for each flowitem when the flowitem first covers the decision point.

Access variables:

current: the current object

item: the item that is covering the decision point

decisionpoint: the number of the decision point that caused this trigger to fire

position: the position of the decision point on the conveyor

Pick Operator

Overview:

This function returns a reference to a dispatcher (or operator) that the operator request will be sent to. The return value must be cast into a number, because the function is hard-coded to return a double.

There are also additional return value options for this picklist:

- If the field returns 0, then the Processor will call no operators.
- If the field returns -1, then the Processor will call no operators, but it will also call this function again at the time that it is ready to release operators. When the function is called again, the Processor will pass PICK_OPERATOR_PROCESS_RELEASE or PICK_OPERATOR_SETUP_RELEASE as the trigger parameter, depending on whether it is the setup or process step. This allows you to explicitly create task sequence(s) for calling operators, and then explicitly release them at the end, all from within the same code field. For an example of this, look at the code for the "Multiple Teams" option in the picklist.

Access variables:

current: the current object

item: the involved flowitem

trigger: the involved trigger. Possible values are PICK_OPERATOR_PROCESS, PICK_OPERATOR_SETUP, PICK_OPERATOR_PROCESS_RELEASE, PICK_OPERATOR_SETUP_RELEASE.

Place in Bay

Overview:

This function returns the bay number in which to place the entering flowitem. Bays are numbered starting at 1 for the bay closest to the Rack's origin and increasing numerically going away from the origin.

Access variables:

current: the current object

item: the involved flowitem

Place in Level

Overview:

This function returns the level number in which to place the entering flowitem. Levels are numbered starting at 1 for the level closest to the Rack's origin and increasing numerically going away from the origin

Access variables:

current: the current object

item: the involved flowitem

Pull Requirement

Overview:

This field is a boolean expression returning either true (1) or false (0). It is evaluated in the OnInOpen event of this object whenever an input port becomes ready. An input port is ready when it is open AND the upstream output port that it is connected to is open. The connection becomes ready when either the input port is opened, OR the upstream output port is opened (assuming the paired port is already open). This Pull Requirement expression will be evaluated for each ready flowitem within the object connected to the input port which just became ready causing the OnInOpen event to fire. If the expression returns a true (1), then the ready flowitem will be pulled in through the ready input port.

Access variables:

current: the current object

item: the ready flowitem for which the Pull Requirement is currently being evaluated

port: the number of the input port that became ready

Pull Strategy

Overview:

When this object is in pull mode and it is ready to receive its next flowitem, it will first evaluate this Pull Strategy field to determine which input port to open. If the Pull Strategy field returns a zero (0) for the input port number, then it will open all of its input ports. When an input port is opened, the OnInOpen event of this object will execute immediately if the connected upstream output port is already open, or the OnInOpen event will fire in the future when the upstream output port is eventually opened. When the OnInOpen event fires, the PullRequirement will be evaluated, determining which flowitem will actually get pulled in.

Access variables:

current: the current object

Rise/Fall Through Mark Triggers

Overview:

This code is executed when the content rises up or falls down through the user defined high, low, or middle mark.

Access variables:

current: the current object

item: the involved flowitem

Send Requirement

Overview:

This field is evaluated on a MergeSort when a flowitem reaches an output position on the conveyor. It should return a true or false (1 or 0) as to whether the flowitem should be released out the respective port.

Access variables:

current: the current object

item: the object that is ready to be sent

port: the output port that the flowitem has arrived at.

Send To Port

Overview:

This field is evaluated once for each flowitem at the time the flowitem is ready to be sent to the next object. In a Processor object, the flowitem is ready to be sent at the end of its processing time. In a Queue, the flowitem is ready to be sent after the batch has accumulated and has been released.

The SendToPort expression must return a valid output port number. The output port will be opened, and the port number will be assigned to the flowitem. The flow item will then be pushed (or pulled) when the port connection becomes ready (output port and connected downstream input port are open). In the special case where the SendToPort expression returns a 0, all output ports will be opened, and the flowitem may leave through the first ready port. If the object is configured to reevaluate sendto, then this field is also evaluated every time a downstream object becomes ready to receive a flowitem. If the function returns a -1, then the flowitem will not be released at all, and should be released later on using the releaseitem() command, or should be moved out using the moveobject command.

Access variables:

current: the current object

item: the object that is ready to be sent

Split Quantity (Split/Unpack Quantity)

Overview:

This function returns the number of items to split/unpack from the current item.

Access variables:

current: the current object

item: the involved object. This is the item that will be split/unpacked

Transport Dispatcher (Request Transport From)

Overview:

This function returns a reference to the dispatcher (or transport) that the transport request will be sent to. The return value needs to be cast into a number since this function is hard-coded to return a double.

Access variables:

current: the current object

item: the item to be transported

TaskExecuters Picklists

1. **Break To (Break To Requirement)**
2. **Load / Unload Time**
3. **Pass To**
4. **Queue Strategy**

Break To (Break To Requirement)

Overview:

Define what type of tasksequences this object will accept during a "break" task in its active tasksequence. A standard tasksequence is made up of a travel - load - break - travel - unload sequence of tasks. This field will only be evaluated when the TaskExecuter performs a "break" task.

When the TaskExecuter receives a "Break" task, this is a notification that it may now put the currently active task sequence back in its queue and see if there are any other task sequences that it might want to do before it finishes the current one. This allows for capabilities such as loading several items before dropping them off. In such a case, the TaskExecuter will first make a check to make sure it is currently capable of "multitasking." This is done by asking "is my content less than my capacity?" If the check is true, then it will execute this code to find a tasksequence to break to. If this code returns NULL, then it will not break at all. If a valid tasksequence numeric pointer is returned, then it will break to the new tasksequence and begin executing it. When finished with new tasks, it will return to the original tasksequence.

Access variables:

tasksequence: a reference to the tasksequence that I'm checking

current: the current object

Load / Unload Time

Overview:

When the TaskExecuter performs a Load or Unload task, it executes the Load / Unload picklist. The value returned is the time in model units that the TaskExecuter will delay before moving on to the next task.

Access variables:

item: a reference to item being loaded / unloaded

current: the current object

station: the FixedResource object currently being loaded from or unloaded to

Pass To

Overview:

This function is fired when the Dipatcher receives a task sequence, and should return the output port that the Dispatcher will pass the tasksequence to. If 0 is returned, the tasksequence will automatically queue up according to the defined Queue Strategy until the tasksequence can be passed to an available Dispatcher or TaskExecuter. If a value greater than 0 is returned, the tasksequence will be sent immediately to the returned port number. If a value of -1 is returned, then the Dispatcher does nothing, but rather assumes all dispatch logic is done within the passto function using the movetasksequence() and dispatchtasksequence() commands.

Access variables:

tasksequence: a reference to the tasksequence node

current: the current object

Queue Strategy

Overview:

When the Dispatcher receives a tasksequence, and the "Pass to" function returned a 0, then when a new tasksequence enters the Dispatcher's queue, the Dispatcher goes through his queue of tasksequences, and executes this same function on each tasksequence, and compares it with the value returned for the tasksequence just received. The new tasksequence will then be sorted from highest to lowest (highest value returned will be sorted to the front of the tasksequence queue) according to the value it returned in this function.

Access variables:

tasksequence: a reference to the tasksequence node

current: the current object

Experimentation Picklists

1. End of Experiment
2. End of Run
3. End of Scenario
4. End of Warmup
5. Performance Measure
6. Start of Experiment
7. Start of Run
8. Start of Scenario

End of Experiment

Overview:

This function is executed once at the end of the last run of the last scenario.

Access variables:

replication: current replication number

scenario: current scenario number

End of Run (End of Replication)

Overview:

This function is executed at the end of each replication run. The replication ends when the model has run out of events or the simulation end time has expired.

Access variables:

replication: current replication number

scenario: current scenario number

End of Scenario

Overview:

This function is executed after the last replication of each scenario.

Access variables:

replication: current replication number

scenario: current scenario number

End of Warmup (End of Warmup Period)

Overview:

This function is executed at the end of the user-defined warmup time for each replication run. The time and system variables are reset, but the flowitems remain where they are.

Access variables:

replication: current replication number

scenario: current scenario number

Performance Measure

Overview:

This function is called at the end of each replication of a scenario, and returns a performance measure value to be recorded for the current replication.

Access variables:

There are no access variables for the performance measure function. Use model() as a starting point.

Start of Experiment

Overview:

This function is executed once at the start of the experiment (before model reset).

Access variables:

replication: current replication number

scenario: current scenario number

Start of Run (Start of Replication)

Overview:

This function is executed at the beginning of each replication run (after model reset).

Access variables:

replication: current replication number

scenario: current scenario number

Start of Scenario

Overview:

This function is executed before the first replication of each new scenario (before model reset).

Access variables:

replication: current replication number

scenario: current scenario number

Other Picklists

1. **Text Display**

Text Display

Overview:

This function is executed every time the VisualTool is redrawn to display 3D text in the model view window.

Access variables:

current: the current object

textnode: the node that will store the text to display (a variable node in the visual tool)

Task Sequences

1. Concepts
    - Custom Built Task Sequences
    - Task Sequence Preempting
    - Coordinated Task Sequences
2. Reference
    - Quick Reference
    - Task Sequence Types
    - Querying Task Sequences

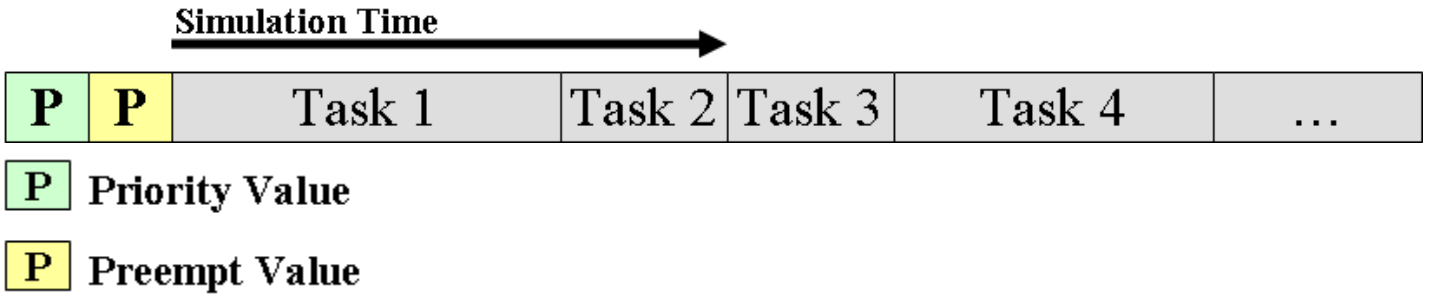
Tutorials

- Task Sequence Tutorial 1
- Task Sequence Tutorial 2
- Task Sequence Tutorial 3

Task Sequences Concepts

What is a Task Sequence?

A task sequence is a series of tasks to be executed in sequential order by a TaskExecuter, as shown in the figure below. The term TaskExecuter implies any object that inherits from the TaskExecuter class. This includes Operators, Transporters, Cranes, ASRSvehicles, Robots, Elevators, and other mobile resource objects. An object is a TaskExecuter if the TaskExecuter tab page options are in its Properties window.



In addition to being a series of tasks, each task sequence has a priority value, which defines the importance of executing that task sequence with respect to other task sequences. Each task sequence also has a preempt value, which defines whether that task sequence should cause other task sequences' execution to be halted in order to execute this task sequence.

Automatically Created Task Sequences

FixedResources have a default mechanism for creating task sequences to move flow items to the next station. You can use this default functionality by checking the "Use Transport" box in the Flow tab page of the FixedResource's Properties. Processors also have a default mechanism for creating task sequences to call operators for setup times, process times and repair operations. This is done by modifying the Processor or ProcessTimes tab page on a Processor, Combiner, or Separator. Each of these default mechanisms triggers a task sequence to be automatically created.

How Task Sequences Work

When you check the "Use Transport" field on a Flow tab page, the following task sequence is created.

1. Travel to the object currently holding the item
2. Load the item from that object
3. Break
4. Travel to the destination object
5. Unload the item to the destination object

When a TaskExecuter executes this task sequence, it will execute each task in order. Each task mentioned above corresponds to a specific task type. Notice in the above example that there are two "Travel" task types in the task sequence, one "Load" task type, one "Unload" task type, and one "Break" task type.

Travel Task

The "Travel" task type tells the TaskExecuter to travel to some object in the model. This may be done in several different ways, depending on the model's setup. If the TaskExecuter is connected to a network,

then the travel task will cause it to travel along the network, arriving at a network node that is connected to the desired destination object. If the TaskExecuter is a Crane object, then it will lift up to a modeler-defined height, then travel to the X/Y location of the destination object. Hence, a travel task can imply several things, depending on the setup of the model, as well as the type of object that is being used. The one thing, however, that all travel type tasks have in common is that they all have some destination object in the model that they are trying to get to.

Load and Unload Tasks

The "Load" and "Unload" task types tell the TaskExecuter to load or unload a flow item into or out of a station. This usually involves traveling an offset distance in order to pick or place the item at the right spot, as well as going through a modeler-defined load or unload time before transferring the item. While the load/unload time is handled in the same way for all TaskExecuters, the offset travel may differ depending on the type of TaskExecuter. A Transporter, for example, will travel to the pick/place location while lifting its fork up to the pick/place height. A robot, on the other hand, will rotate to the spot where the item should be picked/placed. For more detail, see offset travel.

Break Task

The "Break" task type tells the task executer to check if there are any other task sequences that it can "break" to. For example, if a transporter has two items that are waiting to be picked up from the same location, and it has the ability to load two or more items, then the transporter would have two task sequences to do. Both of them would be like the above mentioned task sequence. One is the active task sequence to pick up the first item, and the other task sequence is placed in his task sequence queue to be executed once he finishes the active task. The break task allows the transporter to stop the first task sequence after he has loaded the first item, and begin the second task sequence, which is to travel to the second item's station and load the second item. If the task sequence did not contain the break task, the TaskExecuter would have to finish the first task sequence completely, unloading the first item before being able to load the second item.

Operator Task Sequences

Here's another example of an automatically created task sequence. The Processor object creates this task sequence to request an operator to work at the processing station. The task sequence is described as follows:

1. Travel to the processing station

2. Be utilized until freed from the processing station.

As in the previous example, the first task tells the TaskExecuter to travel to the station. The second task is a new task type not mentioned in the previous example. This is a "Utilize" task type. It tells the TaskExecuter to go into a given state, like "Utilized" or "Processing", and then wait until it is freed from that state. The operator is freed when the freeoperators() command is called. Since the Processor automatically creates this task sequence, it automatically handles freeing the operator as well.

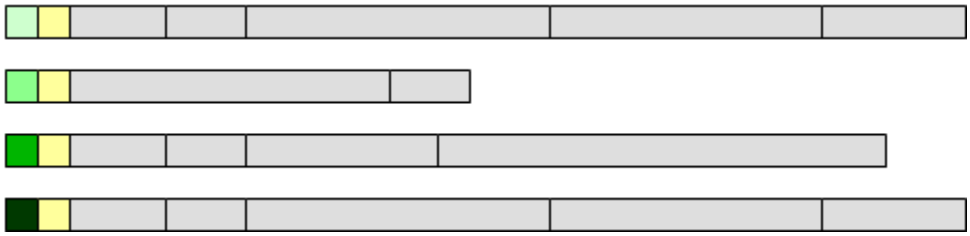
Note: An operator task sequence is not created exactly as described above. In reality there are more tasks added. For simplification purposes, though, we present the above example. For more information on the operator task sequence, refer to the documentation on the requestoperators command in the command summary.

At any given time in a simulation, a TaskExecuter can have one active task sequence as well as a queue of waiting task sequences. A Dispatcher object, on the other hand, can have a queue of waiting task sequences, but cannot actively execute any of those task sequences. Rather, it dispatches its queue of

task sequences to TaskExecuters connected to its output ports. This is what distinguishes the Dispatcher object from the TaskExecuter and its sub-classes.

Active Task Sequence

Queue of Waiting Task Sequences



If no task sequences are preempting, then a TaskExecuter will execute its active task sequence until that task sequence is finished. Then it will make the first task sequence in its queue the active task sequence and start executing that one. This repeats until all task sequences in the queue have been executed.

More Task Types

At this point you have been introduced to 5 task types, namely travel, load, unload, break, and utilize. There are several more task types that can be used when creating task sequences. For a more detailed explanation of each task type and its usage, see Task Types. The next section will address how to create and customize your own task sequences.

Custom Built Task Sequences

You can create custom task sequences using 3 simple commands:

createemptytasksequence()

inserttask()

dispatchtasksequence()

First, create a task sequence by using createemptytasksequence(). Then insert tasks into the task sequence by successive inserttask() commands. Finally dispatch the task sequence with dispatchtasksequence().

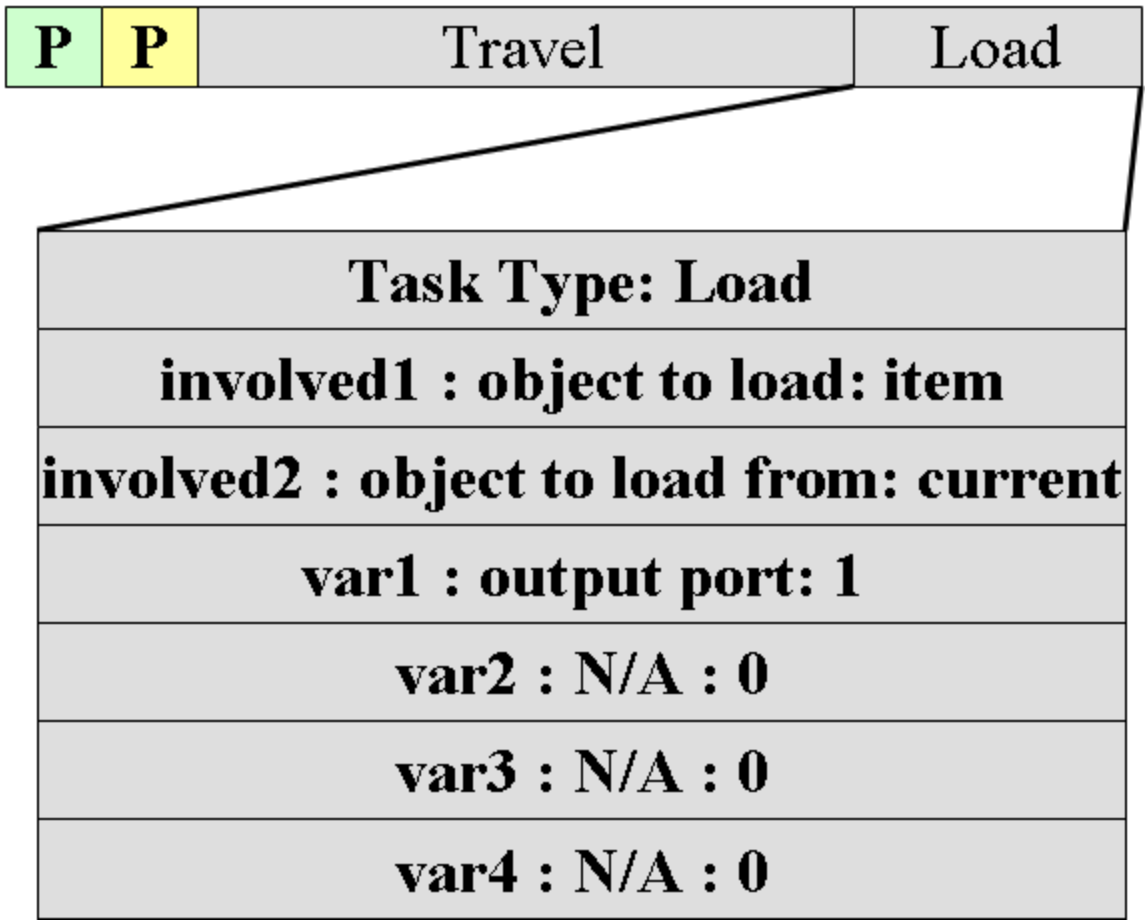
The following example tells a forklift to travel to an object, referenced as "station", then load a flow item, referenced as "item."

```
treenode newtasksequence = createemptytasksequence(forklift, 0 ,0 );
inserttask(newtasksequence, TASKTYPE_TRAVEL, station);
inserttask(newtasksequence, TASKTYPE_LOAD, item, station, 2);
dispatchtasksequence(newtasksequence);
```

If you are confused by the "treenode" syntax, refer the help on the FlexSim tree structure. In brief terms, "treenode newtasksequence" creates a reference, or pointer, to the task sequence as a FlexSim node so that it can be used later when tasks are added to the task sequence.

The createemptytasksequence command takes three parameters. The first parameter is the object that will handle the task sequence. This should be a Dispatcher or TaskExecuter object. The second and third parameters are numbers, and specify the task sequence's priority and preempting values, respectively. The command returns a reference to the task sequence that was created.

The inserttask command inserts a task onto the end of the task sequence. Each task that you insert has several values associated with it. First, it has a type value, which defines what type of task it is. It also has a reference to two involved objects for the task, referred to as involved1 and involved2. These involved objects and what they mean depend upon the task type. For some task types both involved parameters are needed and have meaning, whereas for others, the involved objects are not used. Some task types may use one involved object, and some have involved objects which are optional. Refer to the documentation on task types for information on what a specific task type's involved objects represent. The task can also have up to four number values. These are task variables, referred to as var1, var2, var3, and var4. Again, their meaning depends on the task type. For the load task below, notice that var1 was specified as 1. For a load task, this specifies the output port through which the item will leave the station.



The inserttask command takes two or more parameters, which specify the task's values. The first parameter is a reference to the task sequence into which the task is inserted. The second is the type of task. This can be chosen from an enumerated list of task types. The third and fourth parameters reference the two involved objects. If a specific involved object is not used or is optional for a task type, then you can simply pass NULL into the inserttask command, or even leave that parameter out if there are no number variables that you need to specify either. The fifth through ninth parameters are optional, and define var1-var4. By default, these values are zero.

Note on optional parameters: Even though many of the parameters of the inserttask command are technically optional, depending on the task type, you will still need to specify them. Also, parameters need to still be specified in their correct order. If, for example, you want to specify var1 of the task, but don't care what involved1 or involved2 are, you will still need to pass the NULL value into parameters 3 and 4, even though they are optional, in order to correctly pass var1 in as parameter 5.

Task Sequence Preempting

Every task sequence has a preempting value. Preempting is used to break a TaskExecuter away from its current operation to execute a more important operation. For example, operator A's most important responsibility is to repair machines. When there are no machines to repair, however, it should also transport material throughout the model. If a machine breaks down while operator A is in the middle of transporting a flowitem somewhere, then the operator should stop whatever he is doing and repair the machine, instead of finishing the transport operation. To do this, you will need to use a preempting task sequence to break the operator away from his current operation.

To create a preempting task sequence, specify a non-zero value in the preempt parameter of the createemptytasksequence() command.

createemptytasksequence(operator, 0, PREEMPT_ONLY);

There are four possible preempt values. These values tell the TaskExecuter what to do with the original task sequence(s) that have been preempted.

- 0 - **PREEMPT_NOT** - This value is non-preempting.
- 1 - **PREEMPT_ONLY** - If a task sequence has this value, then the TaskExecuter will preempt the currently active task sequence and put it back in its task sequence queue to be finished later. When a task sequence is preempted, it is automatically placed at the front of the task sequence queue. When the TaskExecuter eventually comes back to the original task sequence, the current task in that task sequence will be done over again, since it was not finished. Also, you can specify a series of tasks to do over again when it comes back to the task sequence using the TASKTYPE_MILESTONE task. This preempt value is the most common used.
- 2 - **PREEMPT_AND_ABORT_ACTIVE** - If a task sequence has this value, then the TaskExecuter will stop the currently active task sequence and destroy it, so that it will never come back to that original task sequence.
- 3 - **PREEMPT_AND_ABORT_ALL** - If a task sequence has this value, then the TaskExecuter will stop the currently active task sequence, destroy it, and destroy all task sequences in its task sequence queue.

To query or change the preempting and/or priority values of a task sequence, you can use the getpreempt(), setpreempt(), getpriority(), and setpriority() commands. For more information on these commands, refer to the Commands documentation found through FlexSim's Help menu.

Interaction Between Multiple Preempting Task Sequences

If a TaskExecuter is currently working on a preempting task sequence, and it receives a new task sequence that is also preempting, it will use the priority value of the task sequence to determine which task sequence to do. If the priority value of the new task sequence is higher than the priority value of the one it is currently working on, the TaskExecuter will preempt its current task sequence and execute the new one. If the priority value of the new task sequence is less than or equal to the priority of the task sequence it is currently working on, then the TaskExecuter will not preempt the active task sequence, but will queue up the new task sequence just like any other task sequence it receives. If it must queue up the task sequence, it will not take the preempt value into account for its queueing logic unless you explicitly tell it to in the queue strategy.

Note on queueing a preempting task sequence: If a preempting task sequence does not actually preempt a TaskExecuter, then it will be queued up like any other task sequence. If you want to have preempting task sequence be brought to the front of the queue, then either make your preempting task sequences have higher priority than all other task sequences, or take preempting into account in the queue strategy.

Preempting Travel Tasks on Networks with Traffic Controls

Preemption can have some undesirable side effects if a TaskExecuter is traveling along a network with TrafficControls when the preemption happens. If preemption occurs when a TaskExecuter is traveling along a network edge, then the TaskExecuter will be "taken off" that edge and connected in an "inactive" travel state (the red line) to the next network node he would have arrived at had he finished traveling the edge. If the network node is a member of a TrafficControl whose area starts at that node (in other words, the TaskExecuter wasn't in the area yet when he was preempted), then the TaskExecuter will be "forced" into the traffic control area, meaning this may cause the number of objects in the area to exceed the maximum for that traffic control area. To avoid this, special logic has been added to the TaskExecuter's preemption mechanism, so that if he is preempted from a travel task directly to another travel task, then instead of being taken off the edge and connected to the next node, the preemption will call redirectnetworktraveler(), which will essentially keep him traveling as he was before on the edge, but his final destination will be changed so that when he arrives at the end of the edge, he will continue on a new path to the new destination. Note that this will only happen if the preemption mechanism can detect that the first durative task that the TaskExecuter will perform after the preemption is another travel task. By durative we mean any task that will take some amount of time to perform: TASKTYPE_TRAVEL, TASKTYPE_LOAD, TASKTYPE_UTILIZE, TASKTYPE_DELAY, etc. are durative, whereas TASKTYPE_SETNODENUM, TASKTYPE_TAG, TASKTYPE_MOVEOBJECT, TASKTYPE_SENDMESSAGE, TASKTYPE_NODEFUNCTION are not. If the preemption detects that the next durative task is not a travel task, then it will take the TaskExecuter off the edge as described above, in which case the TaskExecuter may be forced into a traffic control area.

Preempting With Dispatchers

If a preempting task sequence is given to a Dispatcher, the Dispatcher will not consider the preempt value of the task sequence unless you explicitly tell it to. If the Dispatcher is set to dispatch to the first available TaskExecuter, then it will do just that, and not send the preempting task sequence immediately to a TaskExecuter. If you want the Dispatcher to dispatch preempting task sequences immediately, then you will need to specify such logic in its Pass To function.

A TaskExecuter may be connected back to a Dispatcher by dragging an A-connect from the TaskExecuter to the Dispatcher. If this is done, then when the TaskExecuter receives a preempting task it will pass its current task back to the Dispatcher. The Dispatcher will then redistribute that task according to its dispatching logic. Tasks returned to the Dispatcher in this manner are returned in their current state so that the next TaskExecuter will begin where the previous TaskExecuter left off. This may cause some odd behavior that you should take into consideration when assigning preempting tasks. For example, if a TaskExecuter is traveling with an item when it is preempted, the TaskExecuter that receives this task will perform the travel and unload tasks without ever picking up the item from the first TaskExecuter. Instead, the item will "magically" appear at the right location when the unload completes. In order to prevent these odd behaviors you may want to query the TaskExecuter's state to determine if it is in a "preemptable" state (as you define it) before assigning the new task.

Coordinated Task Sequences

Coordinated task sequences are used for operations which require sophisticated coordination between two or more TaskExecuters. These task sequences implement concepts like allocation and de-allocation of TaskExecuters, as well as synchronizing several operations being done in parallel.

Commands

Coordinated task sequences are built and dispatched using a set of commands which are mutually exclusive from the default task sequence commands. The commands for coordinated task sequences are as follows.

createcoordinatedtasksequence()

insertallocatetask()

insertdeallocatetask()

insertsynctask()

insertproxytask()

dispatchcoordinatedtasksequence()

createcoordinatedtasksequence

The createcoordinatedtasksequence command takes one parameter, namely a reference to an object. This object is designated as the “task coordinator” who holds the task sequence, as well as coordinates the tasks. The task coordinator can also be one of the objects that is allocated within the task sequence. It can be any Dispatcher or TaskExecuter object. Note that selecting a task coordinator doesn't mean allocating that task coordinator. A task coordinator can be coordinating any number of coordinated task sequences at any one time. Also, unlike regular task sequences, coordinated task sequences are not queued up. The task coordinator will start executing the coordinated task sequence immediately when you dispatch it, no matter how many other coordinated task sequences it is coordinating.

insertallocatetask

The insertallocatetask command takes four parameters. The first is the task sequence. Second is the TaskExecuter or Dispatcher to give an “allocated” task to. When the task coordinator gets to an allocate task, it will actually create a separate task sequence with an “allocated” task in it, and pass that task sequence to the specified TaskExecuter or Dispatcher. In the case that it is a dispatcher, meaning you want to allocate any one of several TaskExecuters, then you can use the return value of this command as a key to reference the specific one that gets allocated, since you don’t know exactly which one it is at the time that you build the task sequence. The third and fourth parameters are the priority and preempting values of the separate task sequence that will be created. The fifth parameter is optional, and specifies whether the task is blocking. By default (0), the task is blocking. If 1 is passed in, then the task will not be blocking.

insertproxytask

The insertproxytask command is similar to the inserttask command, with one parameter, the second, added. The second parameter specifies which allocated object you want to do the task. As the task coordinator is the one actually “executing” the task sequence, once he gets to a proxy task, he will instruct the allocated object to do the task “by proxy.” Notice that for involved1 and involved2, you can either pass in a key or a straight reference to an object.

insertsynctask

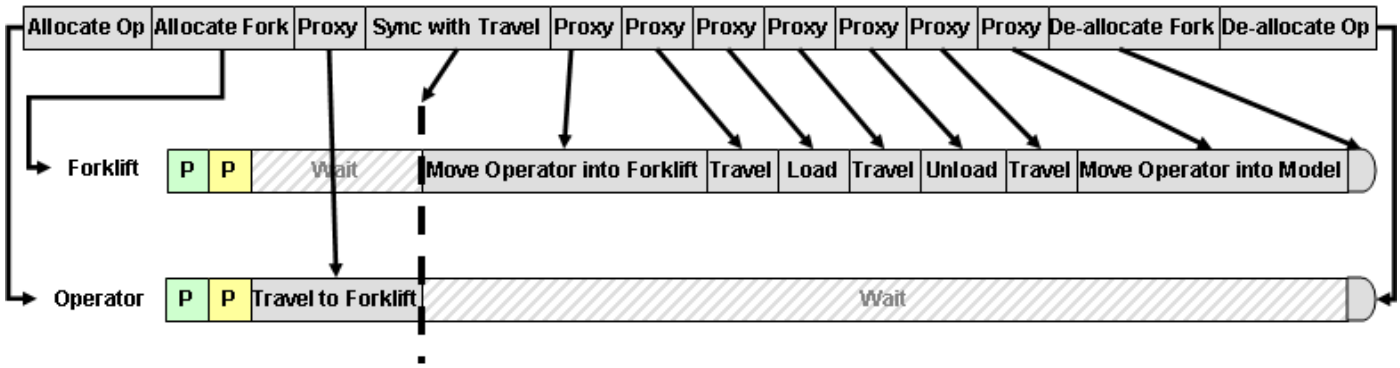
The insertsync task halts execution of the task sequence until a specified task, referenced by its key, is finished. It takes two parameters: the task sequence, and a key value of a given proxy task. It is important to note that proxy tasks which are specified for different TaskExecuters, by default, will be done in parallel, unless a sync task is specified, whereas proxy tasks given to the same TaskExecuter will automatically be done in sequential order, without the need for a sync task.

insertdeallocatetask

The insertdeallocatetask command de-allocates a specific TaskExecuter, referenced by its key. The first parameter references the coordinated task sequence. The second parameter is the allocation key for the resource you want to de-allocate. The third parameter is optional, and specifies whether the task is blocking. By default (0), the task is blocking. If 1 is passed in, then the task will not be blocking.

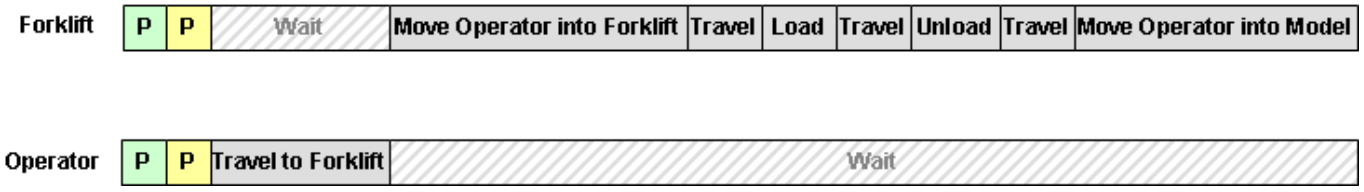
The above code creates a coordinated task sequence that organizes the two task sequences, as shown in the diagram below.

Coordinated Task Sequence



Example

A team of three operators share two forklifts. An operation needs one operator and one forklift. The operator should travel to the forklift, and the forklift should then move the operator into itself. Then the forklift should travel to the load location, pick an item, then travel to an unload location and drop off the item. Then the forklift should travel to its parking location, and unload the operator. Doing this using simple task sequences would be very difficult, because it deals with two different resources that work in a very coordinated fashion. Coordinated task sequences make this example much easier to simulate. The diagram below illustrates the two task sequences that need to be done for the forklift and operator. Notice that there are some parts where one resource needs to wait and do nothing while the other operates.



Code

The code to build the task sequence would be written as follows. It is assumed that references called operatorteam and forkliftteam have been established. These reference dispatchers to three Operator objects, and two Transporter objects, respectively. References have also been established for a loadstation from which to load, an unloadstation to unload to, and the item.

```
treenode ts = createcoordinatedtasksequence(operatorteam);
int opkey = insertallocatetask(ts, operatorteam, 0, 0);
int forkliftkey = insertallocatetask(ts, forkliftteam, 0,0);
int traveltask = insertproxytask(ts, opkey, TASKTYPE_TRAVEL, forkliftkey, NULL);
insertsynctask(ts, traveltask);
insertproxytask(ts, forkliftkey, TASKTYPE_MOVEOBJECT, opkey, forkliftkey);
insertproxytask(ts, forkliftkey, TASKTYPE_TRAVEL, loadstation, NULL);
insertproxytask(ts, forkliftkey, TASKTYPE_LOAD, item, loadstation);
insertproxytask(ts, forkliftkey, TASKTYPE_TRAVEL, unloadstation, NULL);
```

```
insertproxytask(ts, forkliftkey, TASKTYPE_UNLOAD, item, unloadstation);
insertproxytask(ts, forkliftkey, TASKTYPE_TRAVEL, forkliftteam, NULL);
insertproxytask(ts, forkliftkey, TASKTYPE_MOVEOBJECT, opkey, model());
insertdeallocatetask(ts, forkliftkey);
insertdeallocatetaskyts, opkey);
dispatchcoordinatedtasksequence(ts);
```

Note on the above example: There are some model maintenance issues involved here. For example, if you happen to stop and reset the model while the operator is inside of the forklift, you will need to move the operator out of the forklift and back into the model from a reset trigger. Also, whenever you move the operator back into the model, you will need to set its location appropriately, since it is transferring between two different coordinate spaces.

Things to Remember

- The first thing you must do before giving any resource proxy tasks is to allocate that resource.
- You must get the key back from each allocate task, because you will use it later. The insertproxytask command takes a key for the executer of the proxy task. This is the key that the allocation task returns. You also will use this key when de-allocating the object.
- While all proxy tasks for the same allocated resource are executed in sequence, proxy tasks for different allocated resources are executed in parallel, unless you explicitly put blocking tasks in the coordinated task sequence.
- Blocking tasks are ones that block the parallel execution of the coordinated task sequence. The task coordinator goes straight through the task sequence, giving proxy tasks to the appropriate allocated resources, until a blocking task is encountered. It will then wait until that task's blocking requirement is met before continuing the task sequence. In other words, execution of all tasks occurring after that blocking task (regardless of which resource they apply to) will be stopped until the blocking task's requirement is met . The blocking tasks and their blocking requirements are as follows:

    1. Allocation Task: By default this task will block until the specified resource has been allocated. However, if the fifth parameter of insertallocatetask is 1, then the allocate task will not block.

    2. Sync Task: This task will block until the proxy task specified by its key is finished.

    3. De-allocation Task: By default this task will block until the specified resource has finished all its proxy tasks and is de-allocated. However, if the third parameter of insertdeallocatetask is 1, then the de-allocate task will not block.

- The order in which you insert your tasks can have subtle yet important implications. This is especially true for placing your proxy tasks in relation to blocking tasks. Proxy tasks placed after certain blocking tasks can be executed very differently than if those proxy tasks were inserted before the blocking tasks.
- Make sure that you de-allocate all objects that you allocate, or the task sequence won't properly release the objects it has allocated.
- Once you have de-allocated a resource, do not give it any more proxy tasks.

Note on non-blocking de-allocate and allocate tasks: The functionality for allowing these tasks to be non-blocking is still in the beta state. Although we encourage you to use this feature, and there are no known bugs at the time of writing, know that you may run into some problems because this functionality hasn't yet been used extensively.

Task Sequences Quick Reference

Task Type Quick Reference

Task Type	involved1	involved2	var1	var2	var3	var4
TASKTYPE_TRAVEL	destination	NULL	end speed	forcetravel		
TASKTYPE_LOAD	item to load	station	output port	end speed		
TASKTYPE_FRLOAD	item to load	station	output port	end speed		
TASKTYPE_UNLOAD	item to unload	station	input port	end speed		
TASKTYPE_FRUNLOAD	item to unload	station	input port	end speed		
TASKTYPE_UTILIZE	involved	station	state			
TASKTYPE_DELAY	NULL	NULL	time	state		
TASKTYPE_BREAK	send message to	task sequence	check content	check receive	pv(5)* msgp(3)**	pv(6)*
TASKTYPE_CALLSUBTASKS	send message to	task sequence	pv(3)* msgp(1)**	pv(4)* msgp(2)**	pv(5)* msgp(3)**	pv(6)*
TASKTYPE_STOPREQUESTBEGIN	object to stop	NULL	state	repeat	id	priority
TASKTYPE_STOPREQUESTFINISH	object to resume	NULL	repeat	id	already executed	
TASKTYPE_SENDMESSAGE	to object	from object	msgp(1)**	msgp(2)**	msgp(3)**	delay time
TASKTYPE_TRAVELTOLOC	NULL	NULL	x	y	z	end speed
TASKTYPE_TRAVELRELATIVE	NULL	NULL	x	y	z	end speed
TASKTYPE_PICKOFFSET	item	station	x	y	z	end speed
TASKTYPE_PLACEOFFSET	item	station	x	y	z	end speed
TASKTYPE_MOVEOBJECT	object to move	container	output port			
TASKTYPE_DESTROYOBJECT	object to destroy	NULL				

TASKTYPE_SETNODENUM	node to set	NULL	value	increment y/n		
TASKTYPE_TAG	user-defined	user-defined	user-defined	user-defined	user-defined	user-defined
TASKTYPE_MILESTONE	NULL	NULL	range	N/A	N/A	N/A
TASKTYPE_NODEFUNCTION	node	parnode(1)	pv(2)*	pv(3)*	pv(4)*	pv(5)*
TASKTYPE_STARTANIMATION	object	NULL	animationnr	durationtype	durationvalue	
TASKTYPE_STOPANIMATION	object	NULL	animationnr			
TASKTYPE_FREEOPERATORS	object	involved				
Non-user tasks						
TASKTYPE_TE_STOP	NULL	NULL	state			
TASKTYPE_TE_RETURN	task sequence	task				
TASKTYPE_TE_ALLOCATED	coordinator	task sequence				
TASKTYPE_CT_ALLOCATE	dispatcher	allocated object	priority	preempt	front-most proxy task	blocking, (0)yes,(1)no
TASKTYPE_CT_SYNC	NULL	NULL	key or task rank			
TASKTYPE_CT_DEALLOCATE	NULL	NULL	key or task rank	blocking, (0)yes,(1)no		

* pv = parval
** msgp = msgparam

Task Sequence Types

You can also refer to the task type reference guide for quick reference. Task types are as follows.

TASKTYPE_TRAVEL

Here the TaskExecuter travels to the object specified. This is done by making a travel request to the navigator that it is connected to. The navigator then takes over control of the TaskExecuter, and pushes it like a pawn on a chessboard, according to the navigator's own logic, until the TaskExecuter reaches the destination and the navigator notifies the TaskExecuter that the travel task is finished. How the TaskExecuter is pushed is dependent on the type of navigator. If the TaskExecuter is connected to a network, then its associated navigator is a network navigator and will push the TaskExecuter along network paths.

Note on TaskExecuters and navigators: Some objects by default are not connected to a navigator at all. If the TaskExecuter is not connected to a navigator, then it will do nothing for the travel task. The following objects do not connect to any navigators by default: ASRSvehicle, Elevator, Robot, Crane.

involved1	The object to travel to.
involved2	Not used. Use NULL for this parameter.
var1	This specifies the desired end speed for the travel operation. If 0, then the desired end speed will be the maximum speed of the TaskExecuter. A positive value will use that value directly as the end speed. If the end speed is negative, then the functionality is dependent on the navigator's logic. The standard navigators, i.e. the default navigator and the network navigator will interpret negative end speeds as an end speed of 0. Other non-standard navigators may however use a negative end speed value to customize the logic for how to get the TaskExecuter to the destination, allowing the modeler to use different negative end speeds on travel tasks to effect customized travel logic.
var2	Specific to the network navigator. If this value is 1, then the object will travel to the destination node even he is already connected to it.
var3 - **var4**	N/A
Example	`inserttask(ts, TASKTYPE_TRAVEL, outobject(current,1))`

TASKTYPE_LOAD, TASKTYPE_FRLOAD

This task causes the TaskExecuter to load an item from a station. If the TaskExecuter's "Travel Offsets for Load/Unload Tasks" checkbox is checked in its Properties page, then it will travel to the location of the given flow item by querying the location from the station and using offset travel. Then

the TaskExecuter will figure out the load time. At the end of the load time, the TaskExecuter will move the item into itself. For Frload, it will notify the FixedResource right before it moves the item, so that the FixedResource can update its own tracking data. Refer to the FixedResource for more information on when to use Frload and when to use Load

involved1	the object to load (usually a flow item).
involved2	the object to load from (FixedResource is assumed if the tasktype is specified as FRLOAD).
var1	this is the output port through which the object will exit the station. Usually a 0 is fine.
var2	The requested end speed for the task.
var3 - **var4**	N/A

Example	`inserttask(ts, TASKTYPE_LOAD, item, current)` `inserttask(ts, TASKTYPE_FRLOAD, item, current)`

TASKTYPE_UNLOAD, TASKTYPE_FRUNLOAD

This task causes the TaskExecuter to unload an item to a station. If the TaskExecuter's "Travel Offsets for Load/Unload Tasks" checkbox is checked in its Properties page, then it will travel to a drop-off location by querying the station and using offset travel. Then the TaskExecuter will figure out the unload time. At the end of the unload time, the TaskExecuter will move the item into the station. For Frload, it will notify the FixedResource right before it moves the item, so that the FixedResource can update its own tracking data. Refer to the FixedResource for more information on when to use Frunload and when to use Unload.

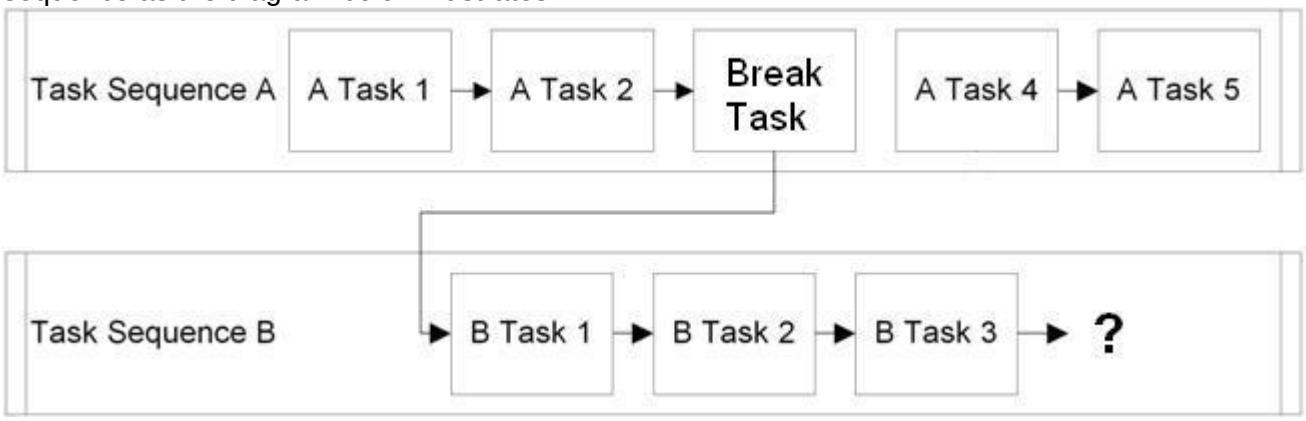
involved1	the object to unload (usually a flow item).
involved2	the object to unload to (FixedResource is assumed if the tasktype is specified as FRLOAD).
var1	this is the input port through which the object will enter the station. Usually a 0 is fine, unless you are unloading to a Combiner, which needs to know the input port an item enters through in order to update its input table.
var2	The requested end speed for the task.
var3 - **var4**	N/A

Example	`inserttask(ts, TASKTYPE_UNLOAD, item, outobject(current, 1))`

	`inserttask(ts, TASKTYPE_FRUNLOAD, item, outobject(current, 1))`

TASKTYPE_BREAK

This task causes the TaskExecuter to "break" from its currently active task sequence to a new task sequence as the diagram below illustrates.



The involved objects and variables allow you to customize how to find the task sequence to break to. In the default case, the TaskExecuter will call its "Break To Requirement" function. This function should return a reference to the task sequence that you want the TaskExecuter to break to. In your break logic you may search through the TaskExecuter's task sequence queue by using task sequence query commands, or you can create the task sequence explicitly using createemptytasksequence. If you don't want the TaskExecuter to break at all, then return NULL.

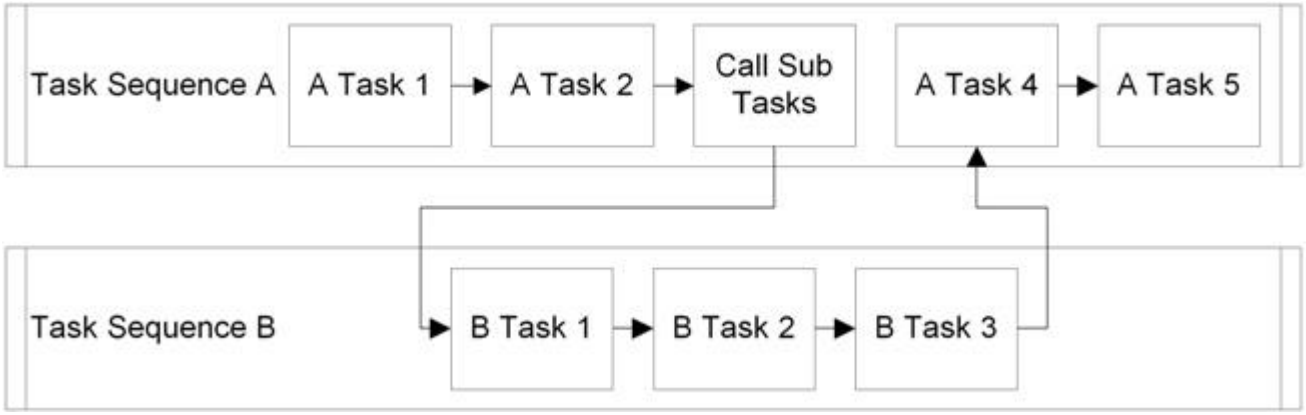
involved1	If involved1 is specified, then it should be a reference to some object in the model. Instead of calling the "Break To Requirement" the TaskExecuter will send a message to this object. The only difference here is the place in which you place your logic for finding a task sequence to break to. By default, the logic executes in the Break To Requirement, but if this parameter is specified, then you will write your logic in a message trigger. Again, the return value of the message should be a reference to the task sequence. You would most likely use this feature if you want to centralize your logic through messages to a central "Model Control Center." **Note on the return value:** In the message you will need to cast the reference to the task sequence into a number, because message triggers return a number type. You can do this by using the tonum() command: tonum(mytasksequence)
involved2	If involved2 is specified, then it is interpreted as a straight reference to the task sequence that the TaskExecuter should break to. This would only be used if you know exactly which task sequence you want to break to at the time that you create the task sequence with the break in it. This parameter is not used very often, because if you know exactly which task sequence to break to when you create a task sequence, then you can just add the tasks into the original task sequence when you create it. **Note on using both involved parameters:** If the involved1 parameter of this task is specified, then involved2 should be NULL. Likewise, if involved2 is specified, then involved1 should be NULL. These parameters are mutually exclusive. You can also just use the default case by specifying both of the involved parameters as NULL.

var1	This parameter specifies whether or not the content of the TaskExecuter should be screened before performing the break task. By default (0), the TaskExecuter will only perform a break task if its current content is less than the maximum content specified in its Properties page. If var1 is not 0, however, then the TaskExecuter will ignore its current content, and perform the break task anyway. This parameter is also passed in as parval(3) if it is to call its Break To Requirement function, and as msgparam(1) if it is to send a message.
var2	This parameter specifies whether or not the TaskExecuter should check to receive task sequences from an upstream Dispatcher. By default (0), the TaskExecuter will see if it has any task sequences in its queue. If the queue is empty, or if all of the task sequences in its queue are task sequences that have already been started and broken out of, then it will open its input ports and receive a task sequence from an upstream dispatcher. However, if var2 is not zero, then the TaskExecuter will not receive anything from an upstream dispatcher before calling its break logic. This parameter is also passed in as parval(4) if it is to call its Break To Requirement function, and as msgparam(2) if it is to send a message.
var3 - **var4**	These parameters are passed into the Break To Requirement as parval(5) and parval(6), and var 3 is passed into the message as msgparam(3) if the task is to send a message.

Example	`inserttask(ts, TASKTYPE_BREAK, NULL, NULL)` //a basic break. the "Break To" Requirement on the TaskExecuter tab will fire `inserttask(ts, TASKTYPE_BREAK, centerobect(current, 1), NULL)` //Sends a message to the referenced object, where your logic will be written in the OnMessage trigger of the object `inserttask(ts, TASKTYPE_BREAK, NULL, specificTaskSequence)` //breaks to a specified task sequence

TASKTYPE_CALLSUBTASKS

This task is just like the break task, except that it ensures that as soon as the second task sequence is finished, it will return immediately to the next task of the original task sequence. The following illustration shows how this works.



As the diagram shows, Task Sequence A comes to a call sub tasks type, upon which it breaks to Task Sequence B. Immediately after Task Sequence B is finished, it returns to the next task of Task Sequence A. Often Task Sequence B won't be created until Task Sequence A actually gets to the

call sub tasks task. This is because often when you create Task Sequence A, you don't know exactly what you want the TaskExecuter to do when he gets to the point of the call sub tasks task, or you don't have an up-front reference to the objects you need. For example, what if you want the TaskExecuter to travel to one part of the model, then load an item, then travel to another part of the model and unload the item, but you won't have a reference to the item that you want to load until the TaskExecuter arrives at the other side of the model. In this case, you want the TaskExecuter to travel to that portion of the model, then figure out which item to load. Here you would use call sub tasks so that you can resolve the reference to the item at the time the TaskExecuter arrives at the load location. Call sub tasks can also be hierarchical. This means that Task Sequence B can also have a call sub tasks type in it. If you decide to create a task sequence when the TaskExecuter gets to the call sub tasks task, then the involved object you call sub tasks on will receive a message. In that object's OnMessage trigger, you will need to create a new task sequence using createemptytasksequence(), and insert tasks using inserttask(), but DO NOT dispatch the task sequence, instead, simply return the reference to your newly created task sequence.

Note on the Break To Requirement function: When the TaskExecuter comes to this task type, by default, he will call his "Break To Requirement" function. He will pass in a 1 as parval(2), so that within the function you can tell that it is a call sub tasks instead of the usual break task.

Note on Coordinated Task Sequences: Using the diagram above, if Task Sequence B is a coordinated task sequence, then the TaskExecuter that executes the call sub tasks task from Task Sequence A must be the first object to be allocated in the Task Sequence B.

involved1	If involved1 is specified, then it should be a reference to some object in the model. Instead of calling the "Break To Requirement" the TaskExecuter will do one of two things. If involved1 is a reference to an object (a node with object data) then the TaskExecuter will send a message from itself to the object specified by the involved1 parameter. If involved1 is a reference to a dll, flexscript, or c++ node (a node with string data) then it will call nodefunction on that node, and it will pass a reference to itself as parnode(1). The only difference here vs. the default case is the place in which you put your logic for finding a task sequence to break to. By default, the logic executes in the Break To Requirement, but if this parameter is specified, then you will write your logic in a message trigger or nodefunction. Again, the return value of the message/nodefunction should be a reference to the task sequence. You would most likely use this feature if you want to centralize your logic through messages to a central "Model Control Center." **Note on the return value:** In the message you will need to cast the reference to the task sequence into a number, because a message trigger returns a number type. You can do this by using the tonum() command: tonum(mytasksequence)
involved2	If involved2 is specified, then it is interpreted as a straight reference to the task sequence that the TaskExecuter should break to. This would only be used if you know exactly which task sequence you want to break to at the time that you create the task sequence with the break in it. This parameter is not used very often, because if you know exactly which task sequence to break to when you create the original task sequence, then you should just add the tasks into the task sequence when you create it. It does, however, allow you to specify different priority and preempting values for different portions of your task sequence, so that if you don't want a certain portion of your task sequence to be preempted, then you can have that portion be a sub-routine task sequence with a different priority than the original task sequence. **Note on using both involved parameters:** If the involved1 parameter of this task is specified, then involved2 should be NULL. Likewise, if involved2 is specified, then

	involved1 should be NULL. These parameters are mutually exclusive. You can also just use the default case by specifying both of the involved parameters as NULL.
var1 - var4	These parameters are passed into the Break To Requirement as parval(3), parval(4), parval(5) and parval(6). When sending a message, var1, var2, and var3 are passed into the message as msgparam(1),msgparam(2) and msgparam(3) respectively. When calling a nodefunction, var1, var2, var3 and var4 are passed in as parval(2), parval(3), parval(4), and parval(5).
Example	`inserttask(ts, TASKTYPE_CALLSUBTASKS, NULL, NULL)` // The "Break To" Requirement on the TaskExecuter tab will fire `inserttask(ts, TASKTYPE_CALLSUBTASKS, centerobect(current, 1), NULL)` // Sends a message to the referenced object, where your logic will be written in the OnMessage trigger of the object `inserttask(ts, TASKTYPE_CALLSUBTASKS, NULL, specificTaskSequence)` // breaks to a specified task sequence

TASKTYPE_UTILIZE

This task causes the TaskExecuter to go into a given state, and then wait until it is freed from that state with the freeoperators() command. This task is used frequently when you want an operator to "do something" at a station, but at the time you create the task sequence you don't know how long it will take to finish whatever the operator is doing. In such a case, use this task type to cause the operator to go into the state you specify, and then free him when he is finished, using the freeoperators() command. This can be done from a trigger like OnProcessFinish or OnSetupFinish, etc. If you know from the outset how long the operator will have to be "doing something", then you can use the delay task instead.

involved1	Often this parameter will be a reference to a flow item, if the operator's job has to do with processing a flow item. Sometimes it references a station, for example in the case that a station goes down, and an operator is called. Here, the operator is working on the station, and not a flow item, so the station would be the involved1 parameter. You can even specify this parameter to be NULL if you like. In more specific terms, this parameter is a key for matching with the freeoperators command. For example, if this parameter is a flowitem, then when the freeoperators command is called, the same flowitem must be passed into the second parameter of the freeoperators command in order for the operator to be freed properly. Often you will use a team of operators, any one of which can do the job you want. In such a case you would give the task sequence to a dispatcher, and the dispatcher would give it to a member of the team. At the time you call freeoperators, you really don't know exactly which operator finally came and worked on your job, so you send the freeoperators command to the dispatcher, and in the freeoperators command, you make the second parameter match the involved1 parameter that you specified for this task. This allows the dispatcher to basically say to his team, "Any of you who are doing a Utilize task whose involved1 parameter is this can now finish that task". This makes it so that the dispatcher can free certain operators from the right tasks without freeing other operators from the wrong tasks.
involved2	This parameter only needs to be specified if it is possible for the operator to be preempted away from his operation. An operator can be preempted away from an operation by a preempting task sequence, or by a stopobject() command, or by a global

	TimeTable or global MTBF table. If the operator is preempted away from a utilize task, then problems can be caused if the freeoperators command is called before he comes back to the utilize task. If freeoperators is called while he is doing something else, then the operator will simply ignore it, thinking it doesn't apply to him. Then, once he comes back to the operation, he will never be freed because the modeling logic thinks that he's already been freed. This involved2 parameter can be used to help alleviate this problem. If involved2 is specified, then it should point to an object in the model that is responsible for freeing the operator. When the operator is preempted, he will call stopobject() on the specified object, which stop the object, and in most cases thus stop the object from calling freeoperators. Once the operator comes back to the utilize task, he will call resumeobject() on the station, and things will resume as normal, and the operator will eventually be freed. If you would like to know more about preempting, refer to its corresponding help section.
var1	This is the state into which the operator will go during the utilize task. If it is 0, then the TaskExecuter will go into STATE_UTILIZE.
var2 - **var4**	N/A
Example	`inserttask(ts, TASKTYPE_UTILIZE, item, NULL, STATE_UTILIZE)`

TASKTYPE_STOPREQUESTBEGIN

This task causes the TaskExecuter to call stopobject() on the involved1 object. Refer to the stopobject() command documentation for more information.

involved1	This parameter specifies the object to call stopobject() on. If NULL, then the TaskExecuter will call stopobject on himself.
involved2	Not used. Use NULL for this parameter.
var1	This is the state to request the stopped object to go into.
var2	This variable is necessary only if the TaskExecuter that executes this task may be preempted. It is also only needed if you are using milestone tasks. If this variable is set to 0, then the task will only be executed once, even if it is within a milestone task's range and the object is preempted within that range. If this variable is set to 1 and the task is within a milestone task's range, then the task will be executed again each time the object is preempted and needs to do the task over again. By default, the value is 0, meaning the task will only be executed once. Note that if it is 0, on the first execution of the command, the TaskExecuter will change the variable to 2 as a flag to not execute it again.
var3	This is the id for the stopobject command
var4	This is the priority of the stopobject command

Example	`inserttask(ts, TASKTYPE_STOPREQUESTBEGIN, current)`

TASKTYPE_STOPREQUESTFINISH

This task causes the TaskExecuter to call resumeobject() on the involved1 object. Refer to the resumeobject() command documentation for more information.

involved1	This parameter specifies the object to call resumeobject() on.
involved2	N/A
var1	This variable is necessary only if the TaskExecuter that executes this task may be preempted. It is also only needed if you are using milestone tasks. If this variable is set to 0, then the task will only be executed once, even if it is within a milestone task's range and the object is preempted within that range. If this variable is set to 1 and the task is within a milestone task's range, then the task will be executed again each time the object is preempted and needs to do the task over again. By default, the value is 0, meaning the task will only be executed once.
var2	This is the id for the resumeobject() command.
var3	N/A
var4	This variable is managed by the TaskExecuter, and tells whether this task has already been executed once.

Example	`inserttask(ts, TASKTYPE_STOPREQUESTFINISH, current)`

TASKTYPE_SENDMESSAGE

This task causes the TaskExecuter to send a message to the involved1 object.

involved1	Involved1 is the object that the message is sent to. If NULL, then a message is sent to the TaskExecuter himself.
involved2	Involved2 specifies msgsendingobject in the message trigger. If NULL, then msgsendingobject is the TaskExecuter himself. Usually this will be NULL, because it is the only way that you can access the TaskExecuter within the message trigger. However, you may want the message to be sent "from" a different object, so you have the option here.
var1	This parameter is passed in as msgparam(1) in the message trigger.

var2	This parameter is passed in as msgparam(2) in the message trigger.
var3	This parameter is passed in as msgparam(3) in the message trigger.
var4	This parameter tells whether the message sent is to be a delayed message. If 0, then the message is sent immediately. If -1, then the message is sent delayed in zero time. Otherwise, the message is sent in the specified number of seconds. You might think that delayed message sending is a bit redundant, because if you want to send a delayed message, why not insert a delay task followed by a regular send message task. There is a subtle difference. Say, for example, you want the TaskExecuter to wait until a certain number of requirements are met, and the only way you can check those requirements is by executing code. The way that you would do this is, when the TaskExecuter gets to the point where he needs to wait for the requirements to be met, he sends a message to some object, and then either does a utilize task, or a stop request begin task. When the other object gets the message, he is responsible for checking if the requirements are met. If they are already met, then he is to immediately call resumeobject() or freeoperators() on the TaskExecuter. Otherwise he must wait until the requirements are met, and then call resumeobject() or freeoperators(). A problem arises, however, when the requirements are already met and he can immediately allow the TaskExecuter to continue. If the message has been sent immediately, then the TaskExecuter hasn't started the utilize of stoprequestbegin task yet. He is still working on the send message task. So the other object can't immediately call freeoperators() or resumeobject() because he must wait until the TaskExecuter finishes the send message task, and goes on to the utilize or stop request begin. Sending a delayed message in 0 time allows the TaskExecuter to do exactly that, and thus allow the other object to immediately free him if the requirements are met.

Example	`inserttask(ts, TASKTYPE_SENDMESSAGE, current, NULL, p1,p2,p3, delay)`

TASKTYPE_DELAY

This task causes the TaskExecuter to go into a given state, and then simply wait for a specified amount of time.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	This is the amount of time that the TaskExecuter will wait in the specified state.
var2	This is the state into which the operator will go during the delay task. If it is 0, then the TaskExecuter will remain in the previous state it was in.
var3	This variable is reserved by the TaskExecuter. Do not set this variable yourself, or at least don't expect it to stay the same as what you specified it to be.

var4	N/A

Example	`inserttask(ts, TASKTYPE_DELAY, NULL, NULL, time, STATE_NUMBER)`

TASKTYPE_MOVEOBJECT

This task tells the TaskExecuter to move a specified object into a specified container. This would be used if you want the TaskExecuter to load/unload a flow item without going through the offset travel or the load/unload time. Also, this could be used if you want a flow item to be moved, but not into or out of the TaskExecuter.

involved1	The object to move.
involved2	The object to move involved1 into.
var1	The output port of the object that involved1 will exit. 0 is usually fine.
var2 - **var4**	N/A

Example	`inserttask(ts, TASKTYPE_MOVEOBJECT, item, outobject(current, 1))`

TASKTYPE_DESTROYOBJECT

This task tells the TaskExecuter to destroy the specified object. Usually this will be done if a flow item is finished in a model, and is ready to go to a sink. You can destroy the flow item explicitly here. You could also use this to destroy labels. Say for example you have a label that acts as a queue of requests. Once a request has been completed, or is ready to be taken out of the queue, you can destroy it.

involved1	The object to destroy.
involved2	Not used. Use NULL for this parameter.
var1 - **var4**	N/A.

Example	`inserttask(ts, TASKTYPE_DESTROYOBJECT, item, NULL)`

TASKTYPE_SETNODENUM

This task causes the TaskExecuter to set the value on a specified node. This would be used if you want to set a variable or label on the object.

involved1	The node to set the value on. This can be something like label(current, "mylabel"). or var_s(current, "maxcontent")
involved2	Not used. Use NULL for this parameter.
var1	The value to set the node to.
var2	This parameter allows you to either set the value on the node, or increment the value on the node. By default (0), it will set the value of the node. If 1, then it will increment the value on the node by var1.
var3 - **var4**	N/A.
Example	`inserttask(ts, TASKTYPE_SETNODENUM, theNode, NULL, 42)`

TASKTYPE_TRAVELTOLOC

This task causes the TaskExecuter to travel to a specified location using offset travel.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	This is the x location to travel to.
var2	This is the y location to travel to.
var3	This is the z location to travel to.
var4	This is the desired end speed.
Example	`inserttask(ts, TASKTYPE_TRAVELTOLOC, NULL, NULL, 3,3,3)`

TASKTYPE_TRAVELRELATIVE

This task causes the TaskExecuter to travel a specified offset using offset travel. This is like the TravelToLoc task, except that instead of traveling to a location, the TaskExecuter offsets from his

current location.	
involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	This is the x offset to travel.
var2	This is the y offset to travel.
var3	This is the z offset to travel.
var4	This is the desired end speed.
Example	`inserttask(ts, TASKTYPE_TRAVELRELATIVE, NULL, NULL, 1,0,0)`

TASKTYPE_PICKOFFSET

This task causes the TaskExecuter to execute part or all of the travel offset involved in a load task. This also allows you to sequence the travel operation that the TaskExecuter does before doing a load. Let's say, for example, that you have a floor storage area that the TaskExecuter is going to pick an item from. The items are organize in bays (x) and rows (y) on the floor. When the TaskExecuter arrives at the floor storage area, instead of traveling straight to the product to load it, you want him to first travel in the x direction to the right bay, then travel the y and z offsets to the location of the item. This task type allows you to do this. When the TaskExecuter arrives at the floor storage area, you can give him a pick offset task in which you tell to only travel the x portion of the offset. Then you can give him the usual load task, and he will do the y and z offsets once he's finished with the x offset. If you give an object a pick offset task to travel all of the offsets, the effect will be to do the complete travel operation of a load task, without actually loading the object at the end.

involved1	Just like in a load task, this is the reference to the item that would be loaded.
involved2	Just like in a load task, this is the reference to the station from which the item would be loaded.
var1 - **var3**	These parameters are usually either a 0 or 1. They correspond respectively to the x, y, and z portions of the offset travel. If 0, then the TaskExecuter will travel none of the corresponding offset. If 1, then the TaskExecuter will travel all of the corresponding offset. You can also have these values be between 0 and 1. A 0.9 would mean that the TaskExecuter would travel 90% of the corresponding offset.
var4	This is the desired end speed.
Example	`inserttask(ts, TASKTYPE_PICKOFFSET, item, current, 1,0,0)`

TASKTYPE_PLACEOFFSET

This task is just like the pick offset task, except that it does part of all of the offset travel involved with an unload task.

involved1	Just like in an unload task, this is the reference to the item that would be loaded.
involved2	Just like in an unload task, this is the reference to the station to which the item would be unloaded.
var1 - **var4**	Same as pick offset task.

Example	`inserttask(ts, TASKTYPE_PLACEOFFSET, item, outobject(current, 1), 1,0,0)`

TASKTYPE_TAG

This task is exclusively for you to use to "tag" your task sequences. Say for example, that you create 5 general types of task sequences in your model. At certain points in the simulation you need to know which general type a certain task sequence is. By inserting a "tag" task as the first task of all task sequences you create, you can then query that task by using the gettaskinvolved() and gettaskvariable() commands.

involved1	For your use.
involved2	For your use.
var1 - **var4**	For your use

Example	`inserttask(ts, TASKTYPE_TAG, current, centerobject(current, 1), 1)`

TASKTYPE_MILESTONE

This task type is only useful for task sequences that may be preempted. It defines a "bookmark" in the task sequence that the TaskExecuter can revert back to if it is preempted away from the task sequence. Normally when a TaskExecuter is preempted away from a task sequence, it will resume at the same spot it was at once it comes back to the task sequence. The milestone task allows you to tell the TaskExecuter to repeat a whole section of tasks if preemption occurs. The task has a defined range of subsequent tasks for which it is responsible. If the TaskExecuter is within that range and is preempted, then it will revert back to the milestone task. If it has passed the milestone's range, then it will go back to the default preemption functionality.

class="note">**Note on coordinated task sequences:** The milestone task will not work as a proxy

task in a coordinated task sequence. If you want to set bookmarks in a coordinated task sequence, then you should insert a CALLSUBTASKS proxy task, and within the subsequent sub-task sequence, you can insert milestone tasks as needed.	
involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	This parameter is the range of the milestone task, defined in number of tasks. For example, if var1 is set to 3, and the milestone task is the 5th task in the task sequence, then if the TaskExecuter is preempted while executing any one of tasks 6, 7 or 8, then it will revert back to the milestone task.
var2 - **var4**	N/A
Example	`inserttask(ts, TASKTYPE_MILESTONE, NULL, NULL, 3)`

TASKTYPE_NODEFUNCTION

This task type will call nodefunction() on the specified node.	
involved1	The node to call nodefunction() on.
involved2	Passed in as parnode(1). If specified as NULL, then when the TaskExecuter executes the task, it will pass a reference to itself as parnode(1).
var1 - var4	These parameters are passed in as parval(2), parval(3), parval(4), and parval(5) in the nodefunction.
Example	`inserttask(ts, TASKTYPE_NODEFUNCTION, node, NULL, 1)`

TASKTYPE_STARTANIMATION

This task type will call startanimation().	
involved1	Passed as parameter 1 to startanimation(). If NULL, the the TaskExecuter will pass a reference to itself.
involved2	Not used. Pass NULL.
var1	Passed as the animation parameter into startanimation(). Defines the rank of the

	animation to start.
var2	Passed as the durationtype parameter into startanimation().
var3	Passed as the durationvalue parameter into startanimation().
var4	Not used. Pass 0.

Example	`inserttask(ts, TASKTYPE_STARTANIMATION, NULL, NULL, 1)`

TASKTYPE_STOPANIMATION

This task type will call stopanimation().

involved1	Passed as parameter 1 to stopanimation(). If NULL, the the TaskExecuter will pass a reference to itself.
involved2	Not used. Pass NULL.
var1	Passed as the animation parameter into startanimation(). Defines the rank of the animation to start.
var2-var4	Not used. Pass 0.

Example	`inserttask(ts, TASKTYPE_STOPANIMATION, NULL, NULL, 1)`

TASKTYPE_FREEOPERATORS

This task type will call freeoperators(). This might be used as an alternative or as a supplement to the coordinated task sequence mechanism. It allows you, as part of one TaskExecuter's task sequence, to free another TaskExecuter from a utilize task.

involved1	Passed as parameter 1 to freeoperators().
involved2	Passed as parameter 2 to freeoperators(). If NULL, the TaskExecuter will pass a reference to itself.
var1-var4	Not used. Pass 0.

Example	`inserttask(ts, TASKTYPE_FREEOPERATORS, centerobject(current, NULL, 1), NULL);`

Note: The following task types are only for reference purposes, you should never insert one of these task types explicitly.

TASKTYPE_TE_STOP

This task is created when you call stopobject on a TaskExecuter. The TaskExecuter creates a preempting task sequence of priority 100000, and inserts this stop task into it.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	The state that the TaskExecuter should go into when he is down.
var2 - **var4**	N/A

TASKTYPE_TE_RETURN

This task is added onto the end of a task sequence that is returned by a call sub tasks task. It ensures that once the task sequence is finished, it will return to the original task sequence

involved1	This parameter points to the original task sequence to return to. If the original task sequence is a coordinated task sequence, then this will point to the task sequence with the TE_ALLOCATED task in it.
involved2	This parameter points to the actual call sub tasks task as a node.
var1- var4	N/A

TASKTYPE_TE_ALLOCATED

This is a special task specifically used for coordinated task sequences. The task tells the TaskExecuter to be allocated meaning when the object comes to this task, it will notify the object coordinating the task sequence, and then simply wait until it is told to do something by that coordinator.

involved1	The object coordinating the task sequence.
involved2	This is a reference to the coordinated task sequence being executed.
var1 - var4	N/A

Coordinated Task Types

Note: The following task types should never be inserted explicitly by the user. They should instead be inserted using the insertallocatetask(), insertsynctask(), and insertdeallocatetask() commands.

TASKTYPE_CT_ALLOCATE

Here the task coordinator will try to allocate some TaskExecuter. It is by done by creating a regular task sequence with one TASKTYPE_TE_ALLOCATED task in it, and giving the task sequence to a specified object. This task blocks the continuation of the task sequence until an object has been allocated.

involved1	This is a reference to a dispatcher to give the task sequence to. It may be a specific TaskExecuter, or, if the object to allocate can one of several possible objects, then it can reference a dispatcher that dispatches to those task executers.
involved2	This is not specified when the task is created, but will be set once the object has been allocated and will reference that object.
var1	The priority value for the allocation
var2	The preempt value for the allocation
var3	This is changed as the task sequence is executed, and is the rank of the front most proxy task that has been given to the allocated resource
var4	This tells whether the task sequence is blocking. Default (0) is blocking, 1 is non-blocking

TASKTYPE_CT_SYNC

Here the task coordinator blocks the continuation of the task sequence until some previously specified task (referenced by rank), is finished.

involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	The rank of the task to sync to.
var2 - var4	N/A

TASKTYPE_CT_DEALLOCATE

Here the task coordinator notifies an object that it can finished its allocated task, and resume to other tasks. The object is specified by the rank of the allocation task that allocated it.	
involved1	Not used. Use NULL for this parameter.
involved2	Not used. Use NULL for this parameter.
var1	The rank of the allocation task that allocated the object.
var2	This variable specifies whether the deallocation task is blocking. Default (0) is blocking. 1 is non-blocking.
var3 - **var4**	N/A

Querying Information on Task Sequences

Once you have dispatched task sequences, you can query and change certain values on those task sequences. The following commands allow you to make such queries and changes.

treenode gettasksequencequeue(treenode dispatcher)

This command returns a reference to the task sequence queue of a dispatcher/taskexecuter object. It can be treated like a regular treenode. Say, for example, I am a dispatcher, and I want to query things on the first task sequence in my queue. I could access the task sequence by:

first(gettasksequencequeue(current))

treenode gettasksequence(treenode dispatcher, int rank)

This command is another way that you can get references to task sequences. Rank is the rank of the task sequence in the task sequence queue. Also, if rank = 0, then it will return a reference to the currently active task sequence for the task executer, or the task sequence that he is doing right now.

treenode gettaskinvolved(treenode tasksequence, int rank, int involvednum)

This command returns a reference to an involved object for a given task in a task sequence. Rank is the rank of the task in the task sequence. Involvednum is a 1 or a 2, and references which involved object. Say, for example, that a TaskExecuter is about to do a load task, but he wants to know the object from whom he is loading the item. In a load task, involved1 is the item, and involved2 is the station from which to load. Let's say also that I know that the load task is the 3rd task in the sequence, and the task sequence is currently the active task sequence to get a reference to the station, I would code:

gettaskinvolved(gettasksequence(current,0), 3, 2)

You'll need to know task types and which involved means what for a given task type, but once you know that, it's easy. Most of it is documented in the begintask() method of the TaskExecuter in the library.

int gettasktype(treenode tasksequence, int rank)

This command returns the task type of a given task. Rank is the rank in the task sequence. You can compare this with macros like TASKTYPE_LOAD, TASKTYPE_TRAVEL...

int getnroftasks(treenode tasksequence)

Returns the number of tasks in the task sequence that have not yet been finished.

int gettotalnroftasks(treenode tasksequence)

Returns the total number of tasks in the task sequence.

int gettaskvariable(treenode tasksequence, int rank, int varnum)

Returns the value of a variable in the task of a tasksequence. Again, rank is the rank of the task in the task sequence. Varnum is a number between 1 and 4, and is the variable number. As in the involved objects, variable numbers and what they mean depend on the task type.

int getpriority(treenode tasksequence)

Returns the priority of a given task sequence.

void setpriority(treenode tasksequence, double newpriority)

Sets the priority of the tasksequence

int getpreempt(treenode tasksequence)

Returns the preempt value for the task sequence. You can compare this with PREEMPT_NOT , PREEMPT_ONLY, PREEMPT_AND_ABORT_ACTIVE, PREEMPT_AND_ABORT_ALL. For information on preempting, go to Task Sequence Preempting.

void setpreempt(treenode tasksequence, int newpreempt)

Sets the preempt value of a task sequence. You would pass into newpreempt one of the previously mentioned macros. For information on preempting, go to Task Sequence Preempting.

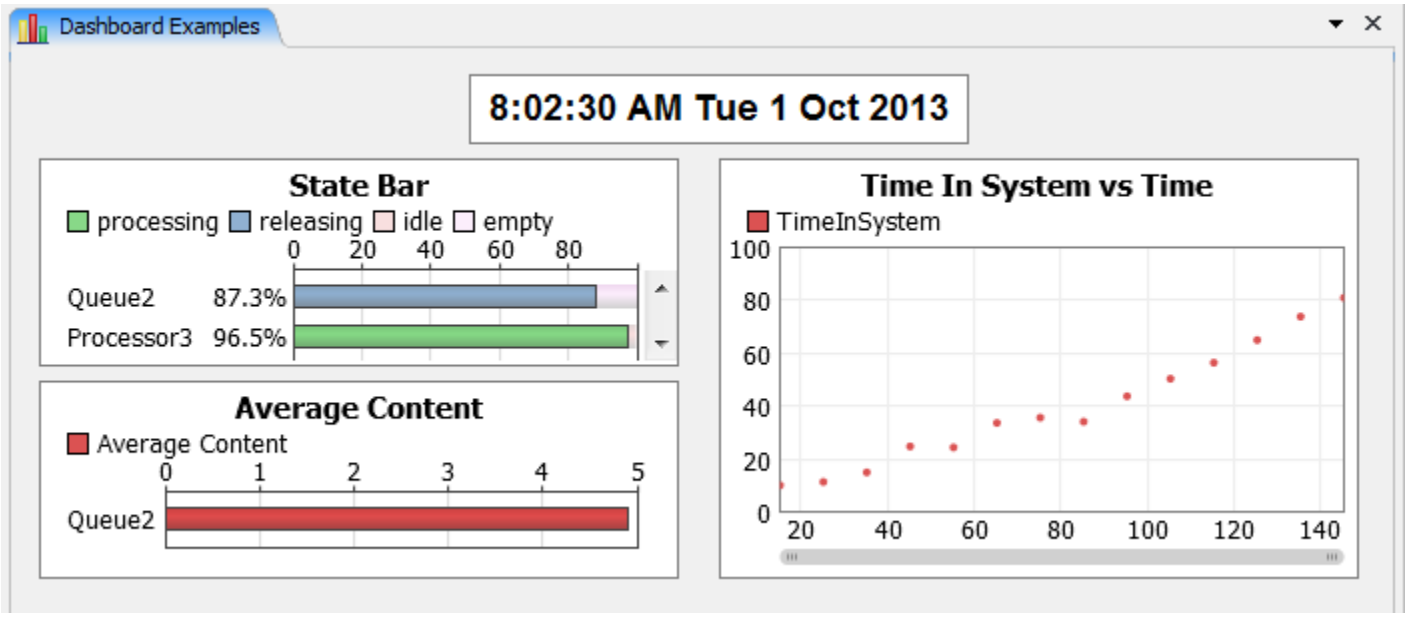
Charting and Reporting

1. **Dashboard**
2. **Reports and Statistics**

Dashboard

1. Concepts
    - Custom Chart
    - Model Input
2. Example
3. Reference
    - Graph Properties
    - Model Input Properties

Dashboard Concepts



Topics

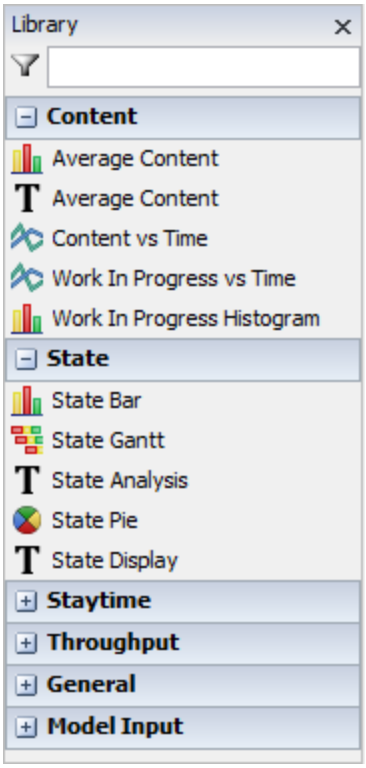
- **Edit Mode**
- **Creating and Arranging Widgets**
- **Exporting Dashboards**
- **Graph Types**
  - **Text Panel**
  - **Bar Chart**
  - **Pie Chart**
  - **Line Graph**
  - **Histogram**
  - **Gantt Chart**
  - **Financial Analysis**
  - **Custom Chart**
  - **Date and Time Display**
  - **Model Documentation**
- **Model Input**

The Dashboard window allows you to view graphs and statistics for the model as it runs. It is especially useful for comparing objects side by side.

Note: Not all statistics make sense for all objects. If a selected object does not have the statistic specified, the graph will not display data for that object.

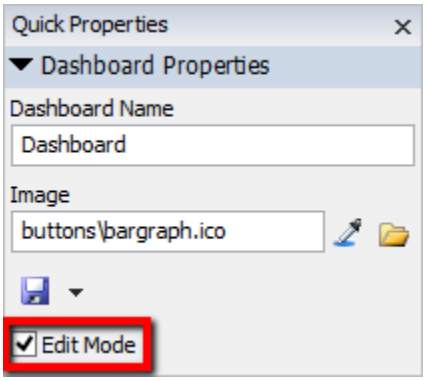
To add a dashboard to your model, go to the **Statistics menu > Dashboard > Add.**

The Library Icon Grid will change to display all of the dashboard widgets that can be created:



Dashboard Graph Types: The list of Dashboard Graphs displayed in the Library Icon Grid are just a starting point, a list of presets. Graphs may be customized to display additional information by editing the Graph's properties.

Edit Mode



When you first create a Dashboard, it will be in edit mode. This allows you to move and resize the Dashboard's widgets. Unchecking the Edit Mode will lock down the widget and allow you to interact with the Model Input objects. The edit mode may also be changed through the right-click menu.

Creating and Arranging Widgets

To add a widget from the Library Icon Grid click and drag the widget to the dashboard window or left-click once on the desired widget to enter creation mode, then left-click again on the Dashboard to create a new widget. Right-click or press the Escape key to exit creation mode.

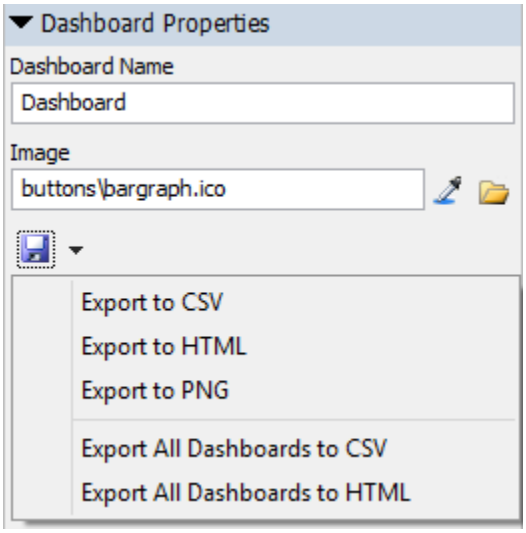
To arrange widgets, left-click on the widget you wish to move. Sizers will appear around the outside of the widget. Click and drag the widget to move it, or click and drag on one of the sizers to resize the widget.

Arrow Keys - Arrow keys may be used to moved Dashboard widgets. Select the desired widget then use the up, down, left or right arrow keys to move the dashboard 1 pixel. Hold the shift key while pressing the arrow keys to move the widget 5 pixels at a time.

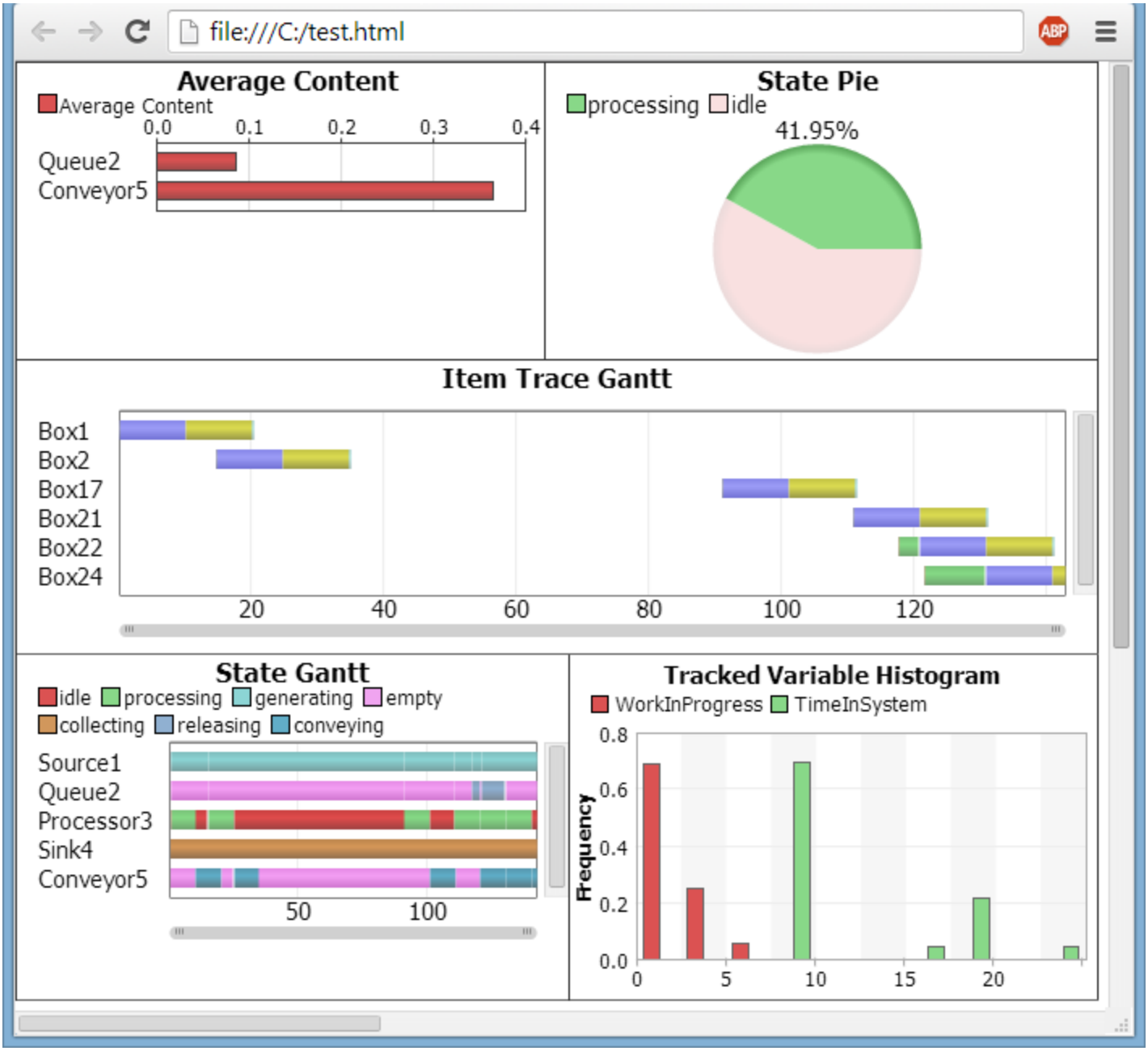
Locking to an Edge

Widget can be "locked" to an edge of the dashboard window by click and dragging a sizer to the desired side of the window. Sides that are locked will show yellow sizers. If a widget is locked to the bottom or right side of the window, that widget will resize as the window is resized. Widgets locked to the top and left side of the window will cause the widget to resize so that the top and left sides of the widget are against the edge of the window, rather than a few pixels off. Once locked, moving a widget will cause it to resize in order to maintain its lock with the edge.

Exporting Dashboards



- Exports dashboard to CSV, HTML and PNG formats. Saving CSV files will only save the Dashboard Data. Saving to HTML will save the Dashboard with its layout to an HTML file that allows you to move, resize and explore the widgets (see image below). Choosing Export All Dashboards to HTML will create one HTML file with all dashboard widgets contained in it. These HTML files are completely self contained and may be opened on any computer with a web browser (though some web browsers may not be supported). After resizing and arranging widgets in the HTML view, the file may be saved again to preserve the layout. Individual widgets may be exported to these formats as well by right-clicking on the widget.



Graph Types

Dashboard Graph Types: The list of Dashboard Graphs displayed in the Library Icon Grid are just a starting point, a list of presets. Graphs may be customized to display additional information by editing the Graph's properties.

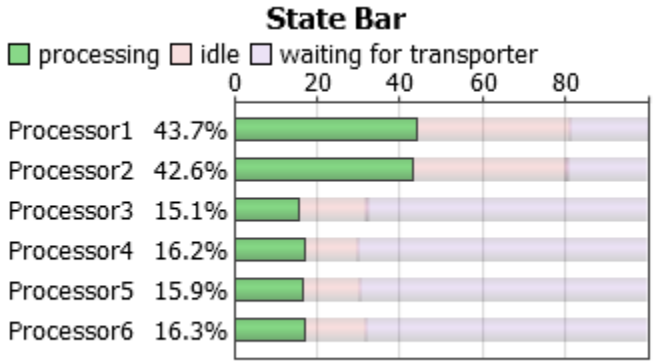
Text Panel

State Analysis

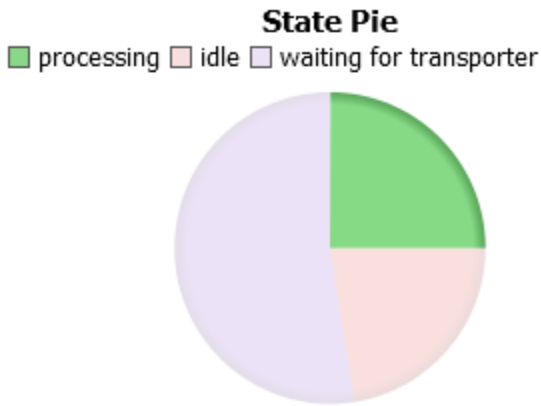
	Total	processing	idle	waiting for transporter
Processor1	43.9%	43.9%	36.5%	19.6%
Processor2	41.3%	41.3%	39%	19.7%
Processor3	15.5%	15.5%	14.1%	70.4%
Processor4	15.7%	15.7%	15.5%	68.8%
Processor5	16.2%	16.2%	15.5%	68.3%
Processor6	16.4%	16.4%	18.6%	65%

Displays text data in a table format. This panel can be used to display state values, statistics and custom data.

Bar Chart

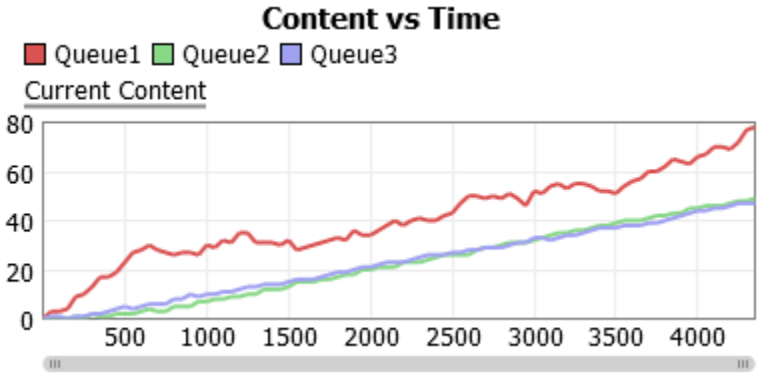


Pie Chart



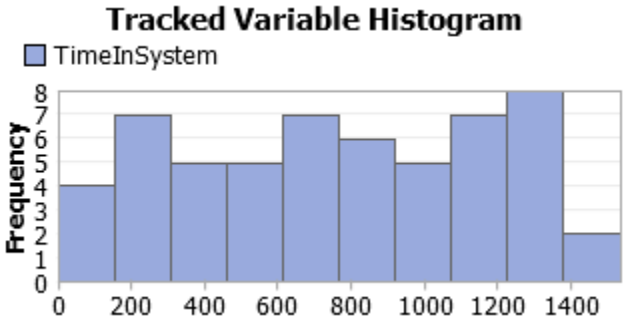
Pie Charts are only available when recording object State data.

Line Graph



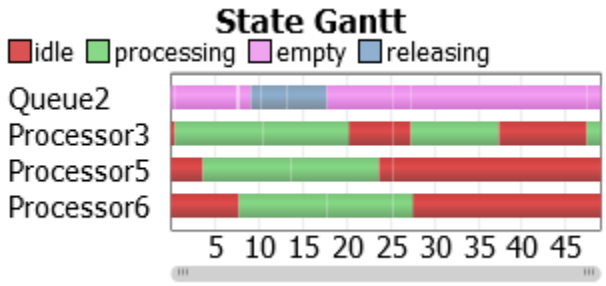
The Line Graph displays data over time. Specify a start time and a time interval between updates.

Histogram



The Histogram is only available for Tracked Variables.

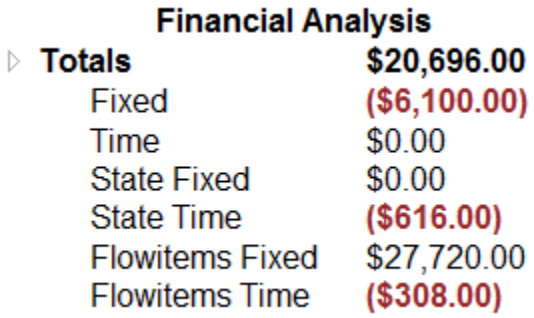
Gantt Chart



There are two types of Gantt Charts:

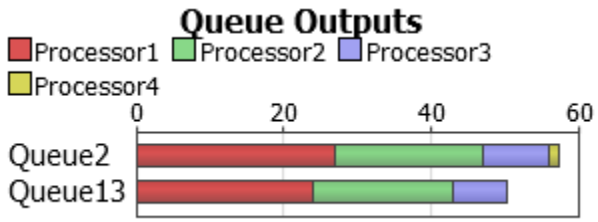
- **State Gantt Chart (displayed above) -** The State Gantt chart shows a time graph of what times and object was in a specific state. State changes in 0 time are not recorded.
- **Item Trace Gantt Chart -** shows what time an item is created and which objects the item travelled through and at what times. Item Trace charts record data even when an item is in an object for 0 time, however, that data is not graphically displayed on the chart. Export the chart's data to CSV to see the fully history.

Financial Analysis



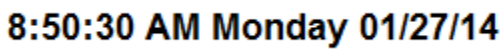
The Financial Analysis graph allows you to specify financial values for objects and flowitems in your model. Individual objects and groups may be added to this graph any number of times. Positive and negative values may be defined for each value. Negative values will be displayed in red surrounded by parentheses.

Custom Chart



The Custom Chart allows you to graph any kind of numeric data in either a table of values, bar chart, or line graph. Rather than adding only objects to this chart, you can add objects, nodes, tables, global variables, bundles etc. For more information, see the Custom Chart page.

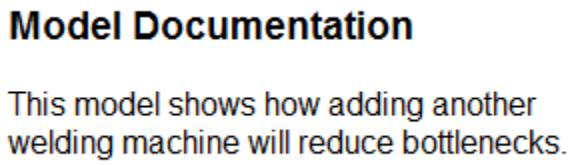
Date and Time Display



The Date and Time Display graph can display the model's current run date and time in multiple formats:

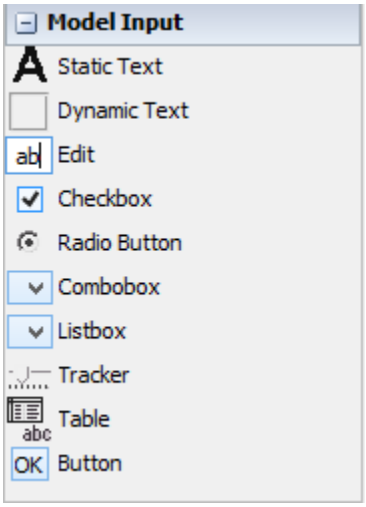
- **No Format -** Displays the current model time beginning from Day 1 at the model start time (as defined in the Model Settings window).
- **Use Default Format -** Displays the current model time using the format defined in the Model Settings window.
- **Custom Format -** This option allows you to define a custom format for the date (displayed above).

Model Documentation



The Model Documentation graph is a custom HTML widget that gives you a blank canvas to add in any custom HTML or flexscript code. Adding flexscript code, you can display any information you want using the pd(), pf(), pt() and pr() commands. These commands will create HTML code to display on the graph.

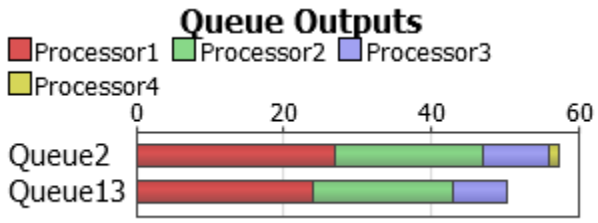
Model Input



Previously, custom user interfaces to control or view variables and parameters from the model was only available through Graphical User Interfaces. Dashboards now have the capability of handling a lot of the same interface requirements as GUIs.

For more information on each ModelInput widget, see the Model Input page.

Custom Chart



The Custom Chart allows you to display data in the dashboard as a table of values, bar chart or line graph. Custom Charts are not limited to objects. Nodes or global variables may be added to the Custom Chart. Code is then executed to specify what data will appear in the chart. This allows the flexibility of displaying just about anything in the Dashboard.

Associations

Custom charts are tied to associations, rather than restricted to only objects as with other Dashboard Widgets. An association can be a node in the tree, a global variable, an object, global table, label table, etc. Associations can be added to the custom chart any number of times, however, the first association is what will be used to specify the number of series to display (see # Series below).

Displaying Data

There are 5 picklists that allow you to specify how many series and how many categories are displayed in the chart. Think of the series as columns in a table, and the categories as rows in a table. All picklists are fired each time the graph is drawn, or for a time graph, each update time. This allows the number of series and categories to be dynamically changed, ie a global table that has columns or rows added to it. A list of presets is available for displaying current values, tables, etc.

Global Variables: Global Variables that are added to the custom chart to be accessed as an association must be either an integer or double type. When using the # Series, # Categories or Data Point Value picklists with global variables, use getnodenum(current) to get the current value of the global variable. This allows you to tie to multiple global variables and other nodes and display all of their data in the same chart. Any other reference to current will give you the global variable's node, ie /Tools/GlobalVariables/myVariable.
Other global variables may be used directly, however, the special functionality of using getnodenum(current) will only return valid data for integer and double types.

Series

The number of series is only calculated once for all associations. This picklist is the first to be fired and passes in the first association object/node. The value returned should be an integer.

Access Variables

- current: the first association object/node in the list.

Series Title

This picklist is fired for each series. These titles will display in the charts legend. The value returned should be a string.

Access Variables

- current: the first association object/node in the list.
- seriesnum: the index of the series.

Categories

The number of categories is calculated for each association in the list. Each category will be a row in the chart, beginning with the first association, in order, down to the last. The value returned should be an integer.

Access Variables

- current: the association object/node.

Category Title

This picklist is fired for each category, for each association. If two or more categories who are sequential (next to each other) have the same title, those categories will become a group. By default, a group's total values are calculated as an average of all of its members. To calculate the total as a sum, the title must have a #sum at the end. ie, *"Total#sum"*. When using groups, the default titles displayed for the group members will be their path in the model. To change this to a custom display, set a Display Name for the association in the Associations tab. The value returned should be a string.

Access Variables

- current: the association object/node.
- categorynum: the index of the category for the association.

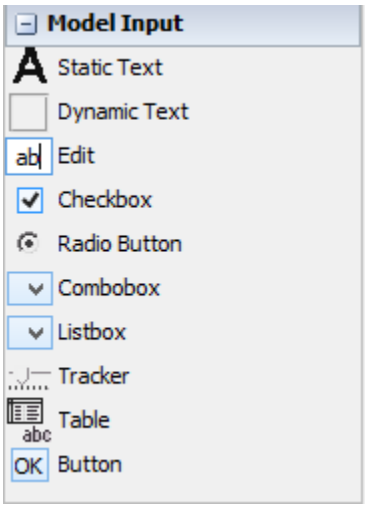
Data Point Value

This picklist is fired for each association, for each category, for each series. The value returned should be a number.

Access Variables

- current: the association object/node.
- seriesnum: the index of the series.
- categorynum: the index of the category for the association.

Model Input



Model Input controls allow you to have interaction between the user and the model without the need for making separate Graphical User Interfaces. Each control has customizable properties that can be set through the Quick Properties window. Each control can have its own ID string that can be used to reference the control using getdashboardcontrol("idName").

Static Text

The Static Text control can be used to either display text, or to display an image.

Dynamic Text

The Dynamic Text control can be linked to a variable, label or node in the model and will display its current value. This can include number data or string data. Dynamic Text is not editable and is for display only.

Edit

The Edit control is similar to the Dynamic Text control but it can also be edited. Changing the value of the Edit will also change the value of the node in the model that it is linked to. ie, linking to the Max Content of a Queue would allow the user to change the Max Content without editing the Queue's parameters directly.

Checkbox

The Checkbox control can be linked to a variable or node in the model. The node will be 'evaluated' as a boolean (true or false, 1 or 0), so linking to a non-number node will have no affect. You can also add code to the Checkbox's OnPress trigger.

Radio Button

The Radio Button control can be linked to a node in the model and it can be linked to other radio buttons. To attach to a group of radio buttons, click on the Link and then click on another radio button in the same Dashboard. This will "Attach" the radio button to the group. Any number of radio buttons may be added to a group. Each radio button in the group has a designated value. If the parent radio button is linked to a node in the model, that node will be set to the value of the selected radio button. You can also add code to the Radio Button's OnPress trigger.

Combobox

The Combobox control can be linked to a variable, label or node with number data. Any number of options can be added to the drop down list. You can also add code to the Combobox's OnSelect trigger.

Listbox

The Listbox control can be linked to a label or node with number or string data in the model. If the node is string data then the name of the listbox item will be set to the node. You can also add code to the Listbox's OnSelect trigger.

Tracker

The Tracker control can be linked to a label or node with number data in the model. Tracker's have a set of additional parameters that can be set including the Minimum, Maximum and Exponential values as well as the Style and if it's a vertical or horizontal tracker. You can also add code to the Tracker's OnDrag trigger.

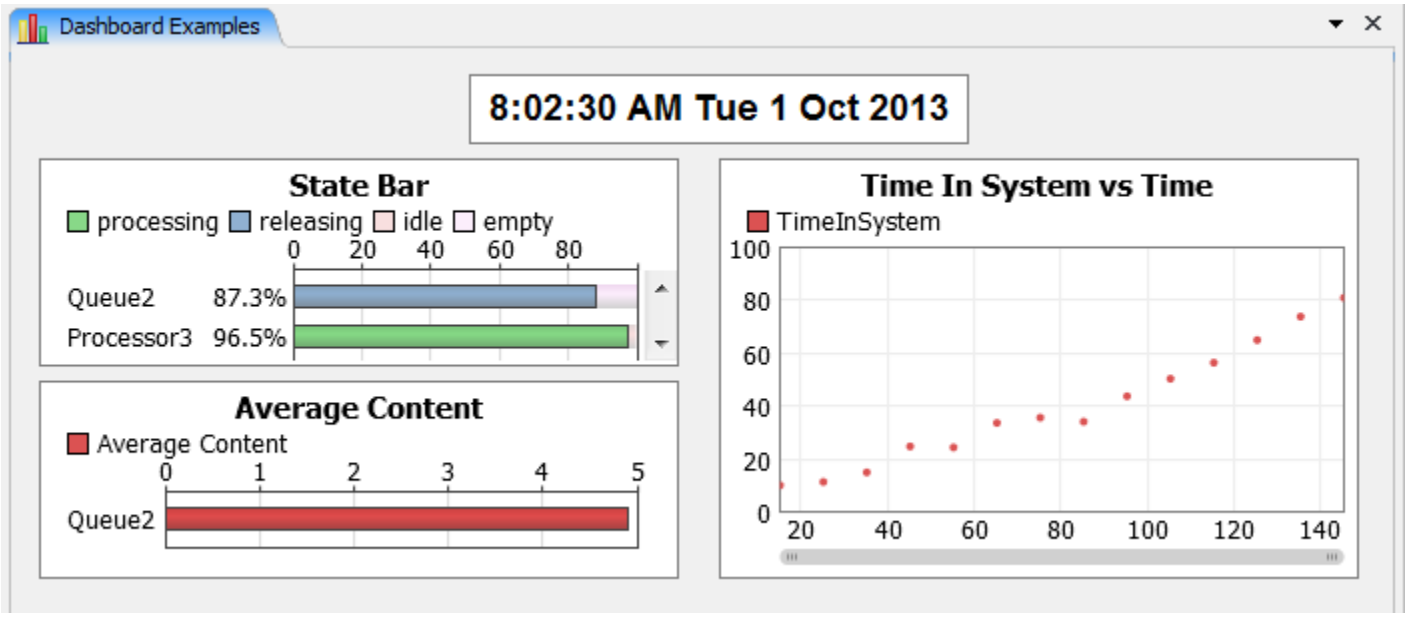
Table

The Table control can be linked to any table in the model. The table will display the linked table's current values. Editing the table will update the linked values directly.

Button

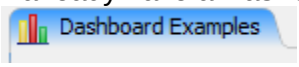
The Button control can be linked to a node in the model. The button can display a text title, or it can display an image. If linked to a node, the button will display the current value of the node. You can also add code to the Button's OnPress trigger.

Dashboard Example

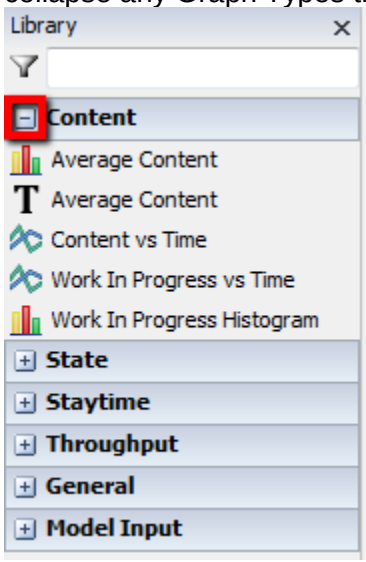


Adding Graphs

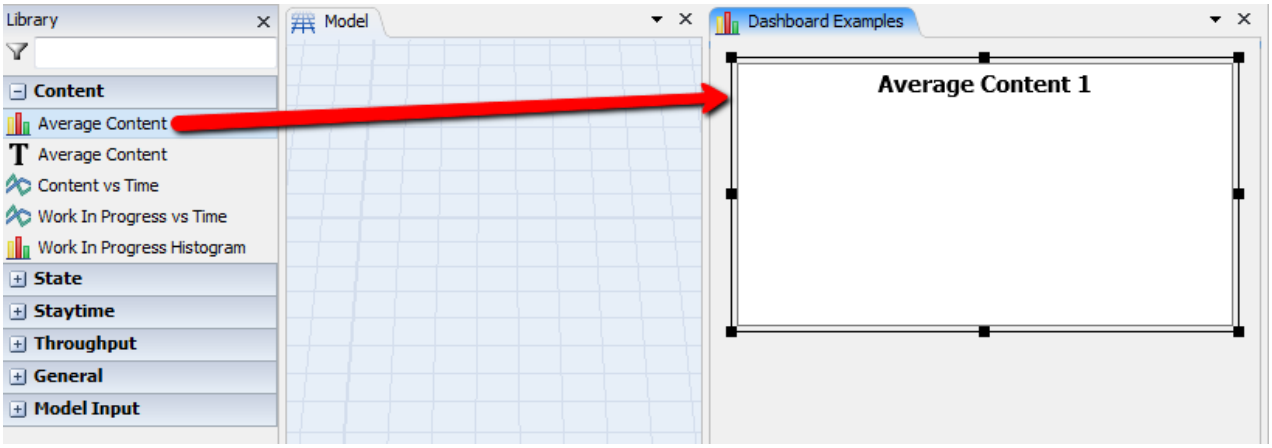
1. Add a Dashboard to the model by selecting **Statistics menu > Dashboard > Add.**
2. If already have a Dashboard displayed, make sure the Dashboard tab is selected



3. If it is not already expanded by default, expand the needed Graph Type from the Library. You can also collapse any Graph Types that are not needed.



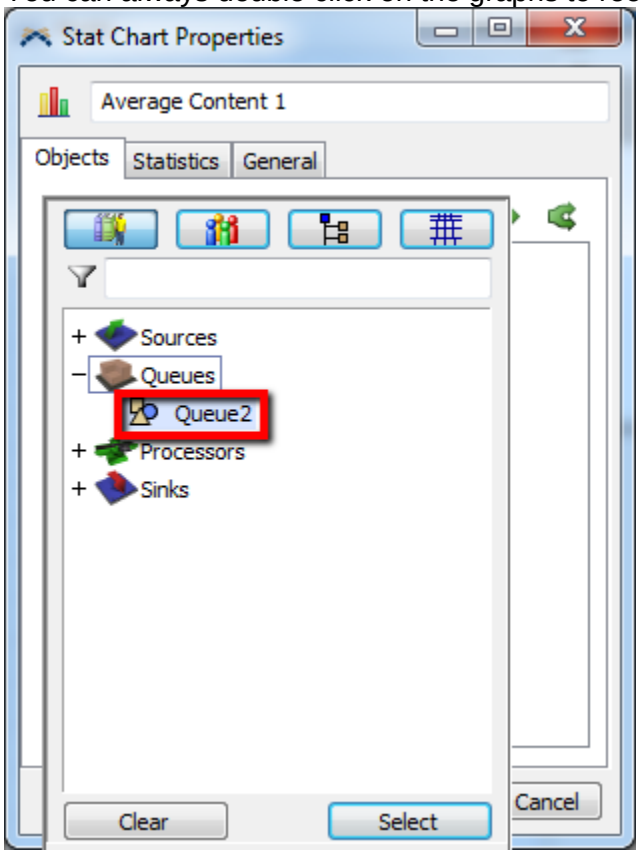
4. **Choose a Statistic**
   Either click and drag one of the options from the menu to the Dashboard or select one of the options and click anywhere in the Dashboard to display the graph.



Model Edit Modes: If you clicked once on a graph in the Library Icon Grid, you will enter a Model Edit Mode. This allows you to create multiple graphs by clicking in the Dashboard view. To get out of Model Edit Mode, right click the view or press the Escape key.

5. **Add Objects**

   The graph properties menu should automatically appear. Click on the ✚ and select which objects to include with this graph. You may select multiple objects through this window if desired. Alternatively, click on the to enter "Sample" mode, then click on an object in the 3D view to add it to your graph. Click on the Select button, then the OK button.
   You can always double click on the graphs to reopen their properties.



6. **Reset and Run the Model**

   The new graph will be blank until the model runs.

Dashboard Reference

Working With Graphs

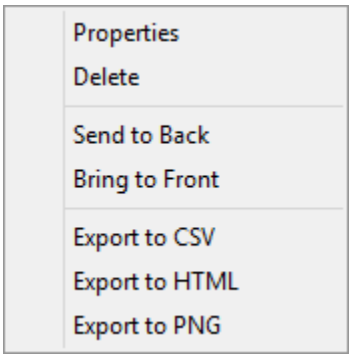
Editing - Double-click the graph to open its properties window. The type of properties window that opens will depend on the type of graph that is being selected. For more information, see Graph Properties

Moving - Click the graph, then drag it by its black edges.

Resizing - Click the graph, then drag any of the black edge sizers.

Auto-resizing - Dragging a graph sizer to the edge of the dashboard window will cause that side to be "locked" to the edge. Locking to the right or bottom edge of the window will cause the graph to resize when the dashboard window is resized.

Context Menu - Right-click the graph to open its context menu.

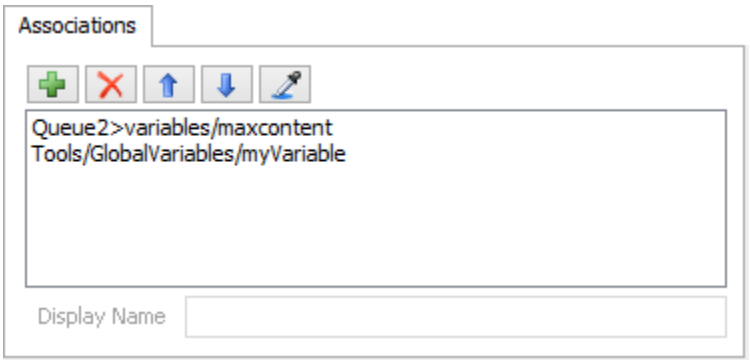


- **Properties** - Opens the graph settings window.
- **Delete** - Deletes the graph from the Dashboard.
- **Send to Back** - Sends the selected graph to the back.
- **Bring to Front** - Brings the selected graph to the front.
- **Export to CSV** - Allows you to save the graph's current data set as a .csv file.
- **Export to HTML** - Allows you to save an self contained HTML file of the graph's current data.
- **Export to PNG** - Allows you to save a .png image file of the graph's current display.

Dashboard Graphs

1. Associations
2. Colors Page
3. Data
4. Date and Time Display
5. Financial Objects Page
6. General Pages
7. HTML Statistic (Model Documentation)
8. Item Trace Page
9. Objects Page
10. Statistics Page
11. Tracked Variable
12. Utilization Analysis Page

Associations Page



- Adds associations to the current list of associations. See the Adding Associations section for more info.

- Removes the selected associations from the list.

- Moves the selected associations up in the list

- Moves the selected associations down in the list

- Sample a node or object in the model.

Display Name - When using groups, if no display name is specified, the titles displayed for the group members will be their path in the model.

Adding Associations

There are multiple ways to add associations to the custom chart.

Tree Browse Dialog

For more information on the tree browse dialog, see the tree browse dialog page.

Global Variables / Tables

The list of Global Variables and Global Tables will be populated based upon what is available in your model. The list of global variables will only include integer and double types.

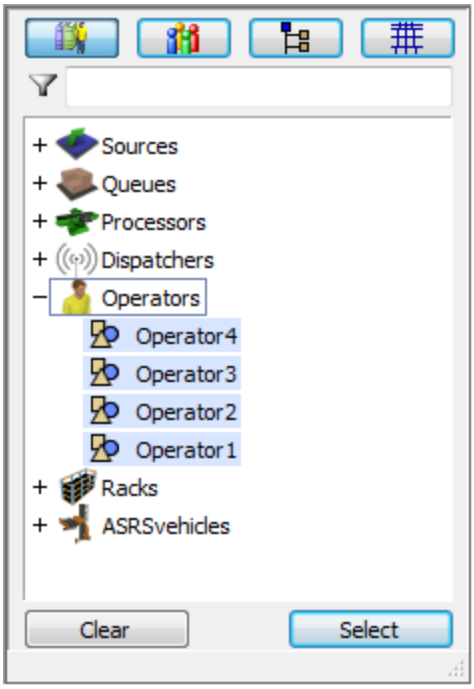
Sampler

For more information on using the Sampler to select objects and nodes, see the Sampler page.

Object Selection Window

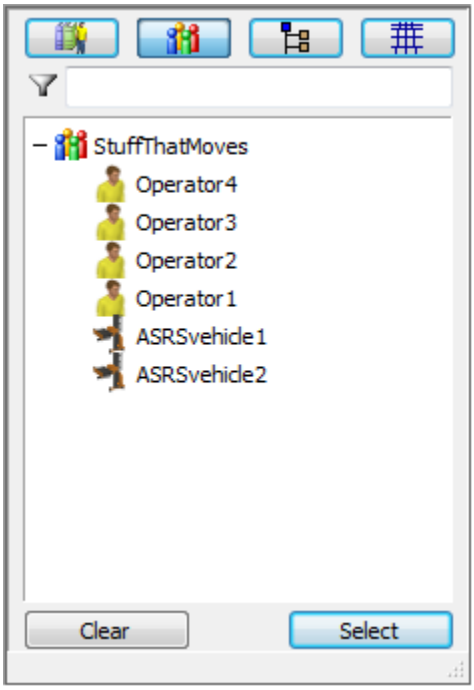
The current mode is highlighted at the top of the browsing window.

Browse by Class



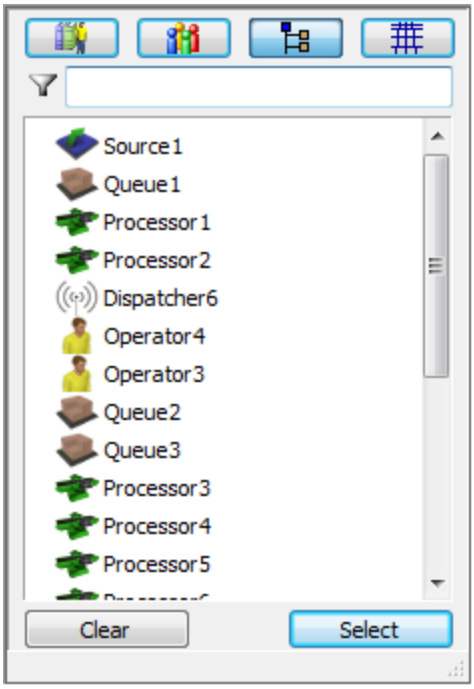
This method of adding objects sorts the objects by class. To select an entire class, click on the type icon. Click on an object to select or deselect it.

Browse by Group



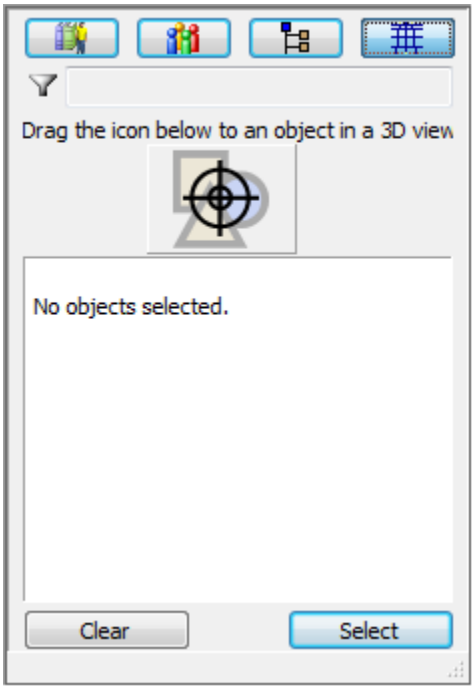
This method of adding objects sorts the objects by group. To select an entire group, click on the type icon. Click on an object to select or deselect it.

Browse by Object



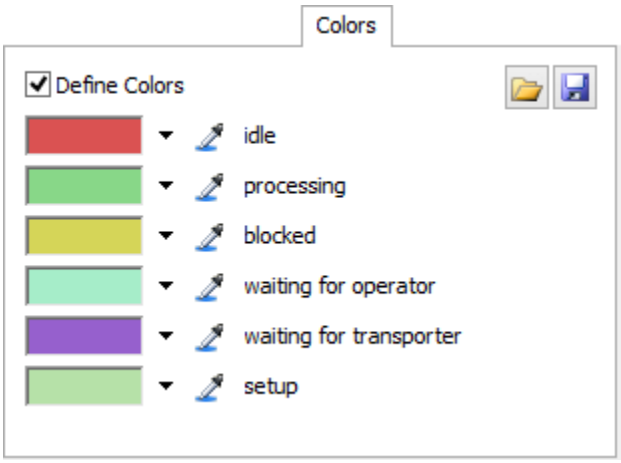
This method of adding objects lists all objects in the model. Click on an object to select or deselect it.

Select by Dragging



This method of adding objects uses the 3D model view. Simply drag the target icon from the current window to an object in the model you would like to add.

Colors Page



The colors page allows you to define colors for your graph. The options available are based upon the graph. For State charts, each State has a color. For statistic graphs (Avg. Content, etc), the colors are for each statistic. Line charts define colors for each object or group. Gantt Charts either display colors for States or for objects.

Colors are not available for Text graphs.

Define Colors - Check to define a custom set of colors.

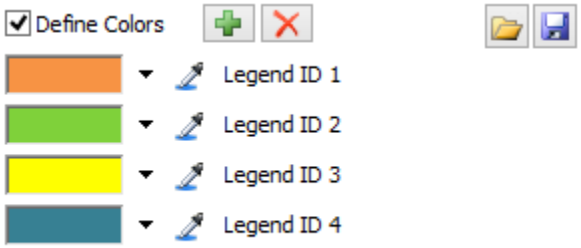
- Loads saved color schemes and allows you to edit color schemes in the tree.

- Saves the current color scheme.

- Displays a color palette.

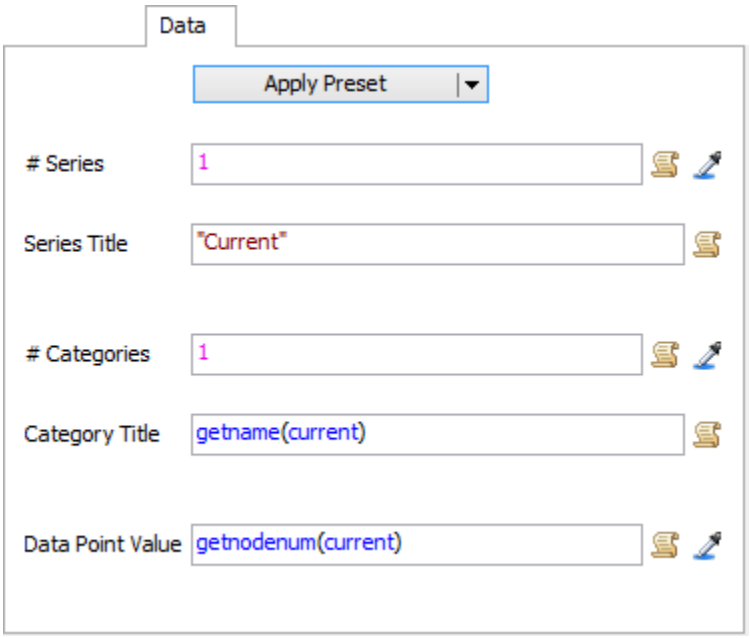
- Allows you to sample a color.

Item Trace Gantt and Custom Chart



The Item Trace Gantt Chart and the Custom Chart dynamically add Legends or Series to their charts so a and are available to add and remove colors.

Data Page

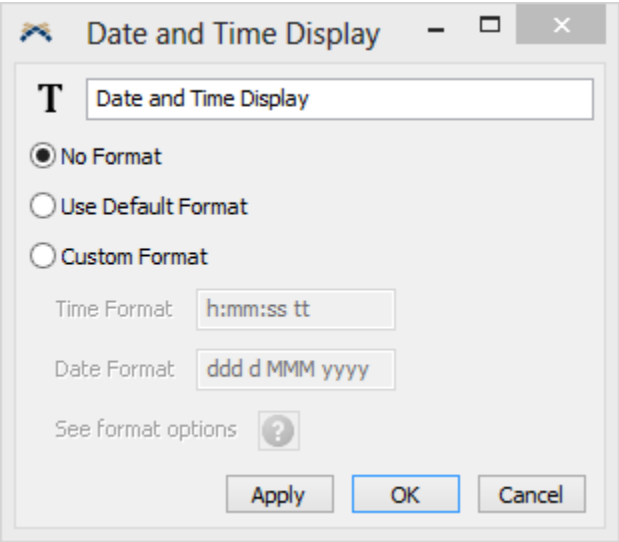


The data table is used by the Custom Chart. There are five picklists that allow you to customize the data to be displayed in a table of values, bar chart or line graph.

Apply Preset - This button contains a list of presets to get you started on using the custom chart. The presets include code to access current node values, table values, bundle values, and getting object input/output values.

See the Custom Chart page for more information on the picklists.

Date and Time Display



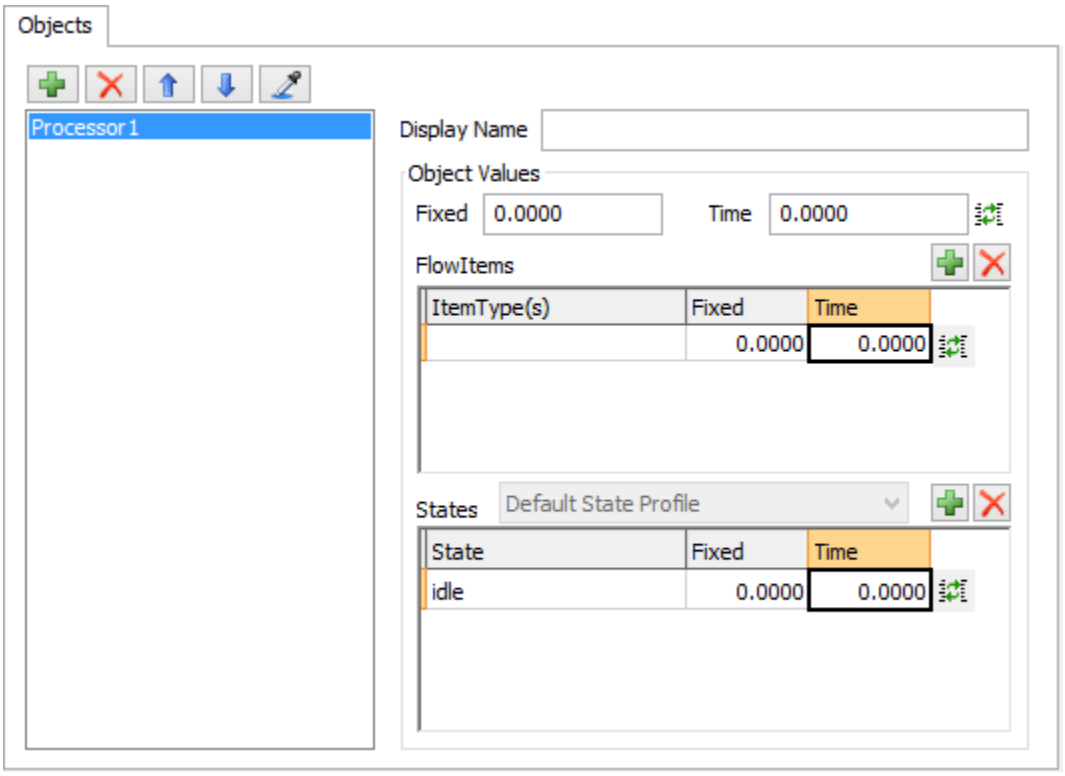
Name Field - Changes the name of the widget to the specified name.

No Format - Displays the current elapsed model time from Day 1 at the time specified in the Model Settings window with the format: **Day 1, 08:00:00**

Use Default Format - Displays the current model date and time as defined in the Model Settings window.

Custom Format - Displays the current model date and time using a custom format.

Financial Objects Page



- Adds objects/groups to the current list of objects. Objects and groups may be added multiple times.

- Removes objects/groups from the current list of objects

- Moves the selected object up in the list

- Moves the selected object down in the list

- Sample an object in the model.

Display Name - Enter text to display on the graph. Leave this field blank to display the object or group name.

Object Values

All object values can be negative or positive numbers. Negative numbers are shown in red and surrounded by parentheses.

Fixed - This value is added once when the model is reset. This value might represent the initial purchase price of a resource.

Time - This value is continually added as the model runs. A value of 1.0 would mean at time 50.0 the total time value for the object would be 50.0.

Use the to convert units to model time units.

FlowItems - Use the and to add or remove items from the table. Each item has an ItemType(s), Fixed and Time. To apply the values to all ItemTypes, leave the field blank. To define multiple ItemTypes, separate numbers using commas and dashes. For example: 1,2,5,10-15,20

- Fixed value is applied when a FlowItem of the specified ItemType enters the object.
- Time value is applied continually while the flowitem is in the object.

Use the to convert units to model time units.

States - Use the and to add or remove items from the table. Each item has a State, Fixed and Time.

- Fixed value is applied when the object enters the specified state. If an object is in the state for 0 time, the fixed value will NOT be applied.
- Time value is applied continually while the object is in the specified state.

Use the to convert units to model time units.

Groups

Unlike other Dashboard graphs, the Financial Analysis graph does not allow you to combine objects together. However, if you have created a group within your model, that group may be added to this graph. All values entered for a group will be calculated for each object contained in the group. For example, if the group had 5 objects in it, and the Fixed value for the group was 20, the total Fixed value will be 100.

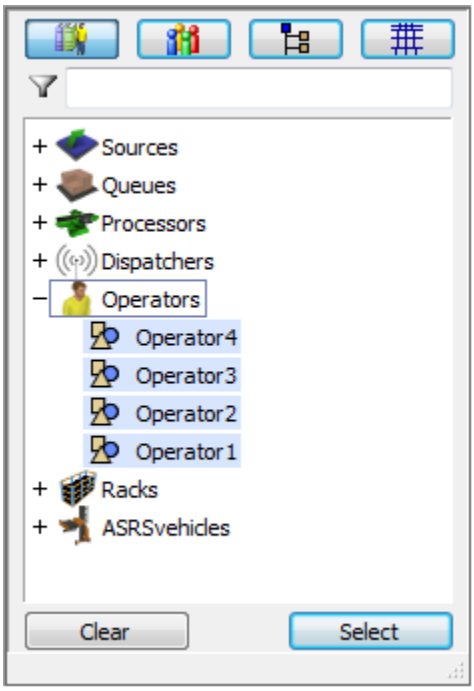
To learn more about groups in FlexSim, see the Groups window.

Adding Objects

There are five ways to add objects to a graph. For more information on using the Sampler to select objects, see the Sampler page.

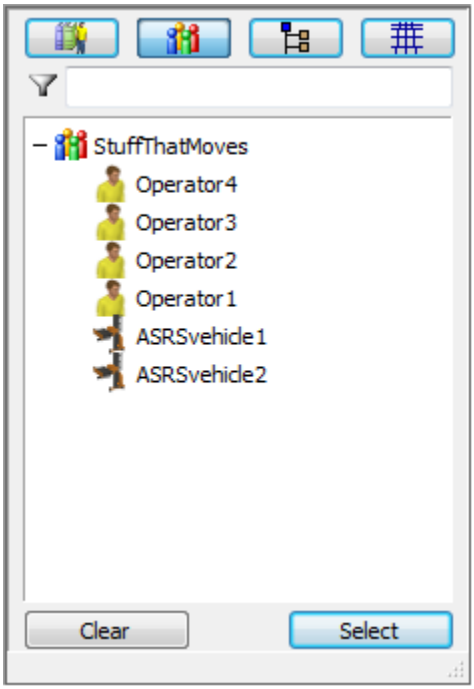
The current mode is highlighted at the top of the browsing window.

Browse by Class



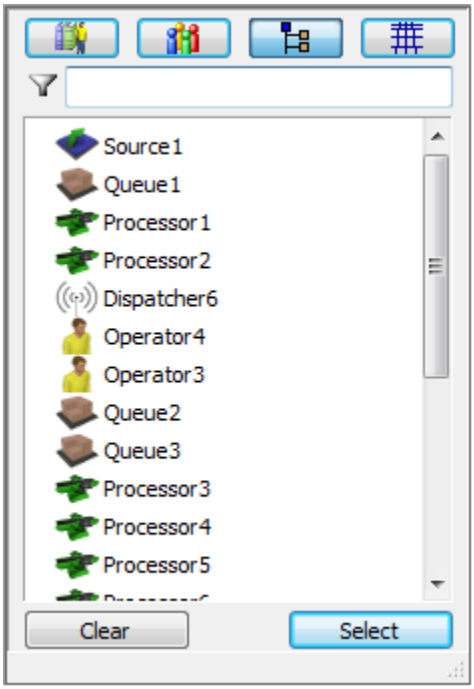
This method of adding objects sorts the objects by class. To select an entire class, click on the type icon. Click on an object to select or deselect it.

Browse by Group



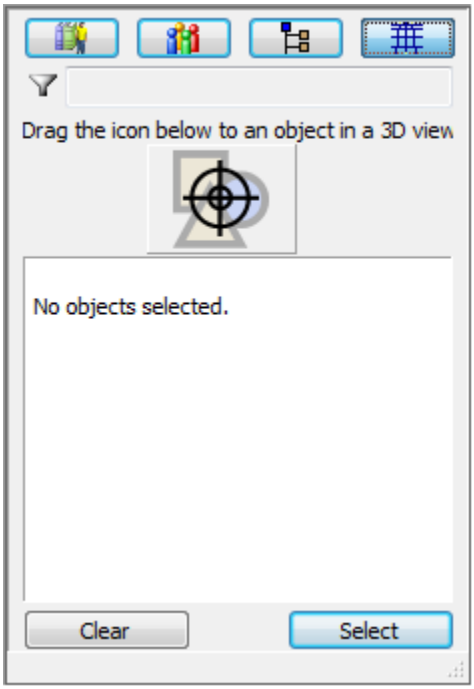
This method of adding objects sorts the objects by group. To select an entire group, click on the type icon. Click on an object to select or deselect it.

Browse by Object



This method of adding objects lists all objects in the model. Click on an object to select or deselect it.

Select by Dragging



This method of adding objects uses the 3D model view. Simply drag the target icon from the current window to an object in the model you would like to add.

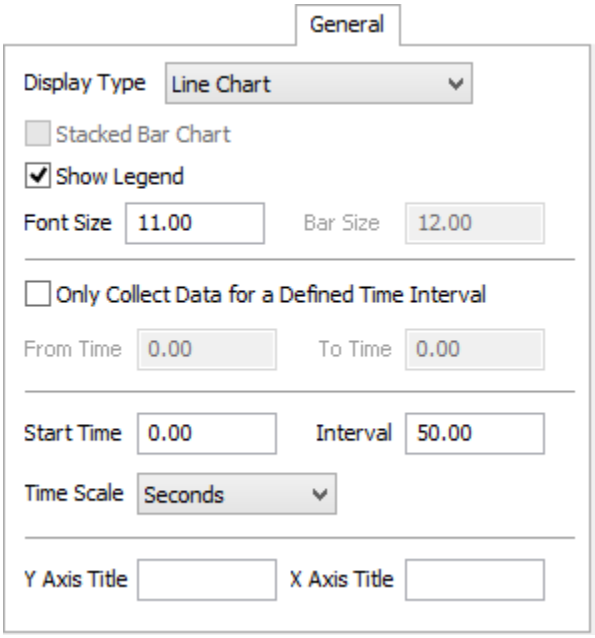
General Pages

The General page of Dashboard graphs vary depending on the statistic object and upon the display type.

Graph Types

- **Line Chart**
- **State Statistic Graphs**
- **Financial Analysis**
- **Gantt Chart**
- **Tracked Variable Graphs**

Line Chart



Display Type - This option changes the display style of graph.

Stacked Bar Chart - Bar Chart Only. This option stacks each segment of the data for one object into a single bar.

Show Legend - This option adds a legend to the graph.

Font Size - This defines the font size of graph text.

Bar Size - Bar Chart Only. This defines the bar height for Bar Charts.

Only Collect Data for a Defined Time Interval - This option creates a time period during which statistics for this graph will be recorded.

From Time - This defines when statistics will start being recorded.

To Time - This defines when statistics will no longer be recorded.

Start Time - This defines the earliest time displayed on a line graph.

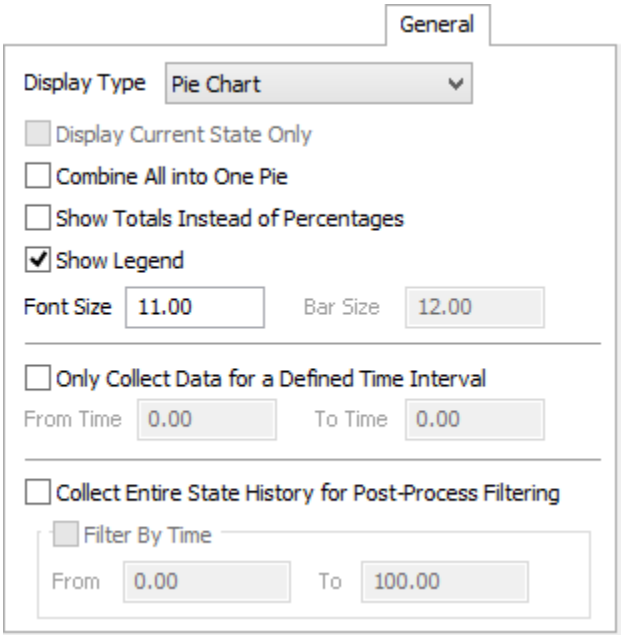
Interval - This defines how often a line graph will update.

Time Scale - This defines the time scale that numbers will be displayed in along the x-axis.

Y Axis Title - Text that will display along the Y Axis.

X Axis Title - Text that will display along the X Axis.

State Statistic Graphs



Display Type - This option changes the display style of graph.

Display Current State Only - This option is only availbe for Table of Values charts. Displays the current state name of objects.

Combine All into One Pie - This option combines all pie charts into a single pie chart. It averages the values from all objects.

Show Totals Instead of Percentages - This option changes the displayed values from percentages to overall totals.

Show Legend - This option adds a legend to the graph.

Font Size - This defines the font size of graph text.

Bar Size - Bar Chart Only. This defines the bar height for Bar Charts.

Only Collect Data for a Defined Time Interval - This option creates a time period during which statistics for this graph will be recorded.

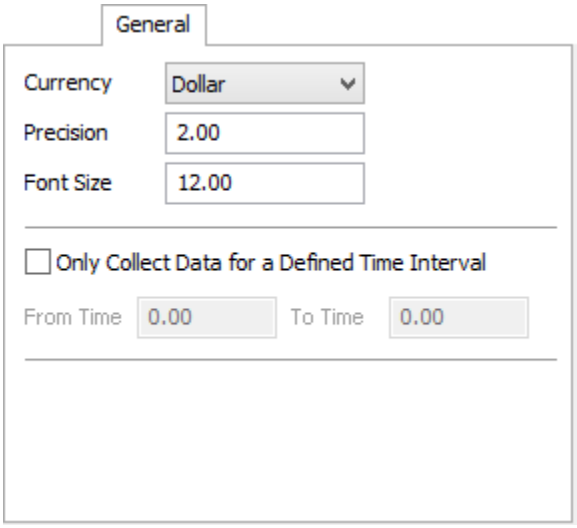
From Time - This defines when statistics will start being recorded.

To Time - This defines when statistics will no longer be recorded.

Post-Process Filtering

If the **Collect Entire State History for Post-Process Filtering** option is checked, FlexSim will record when each state change occurred for every object in the graph. After running the model, you can check the **Filter By Time** option and enter a time interval in the **From** and **To** fields. If you then click **Apply** or **Update**, the graph will display the state data for just that time interval. The **Filter By Time** option is unchecked when you **Reset** the model.

Financial Analysis



Currency - This options changes the prefix of each value to be displayed. The *Other* option allows you to define a custom string value, or leave the field blank to display no prefix.

Precision - This defines how many numbers will be displayed right of the decimal point.

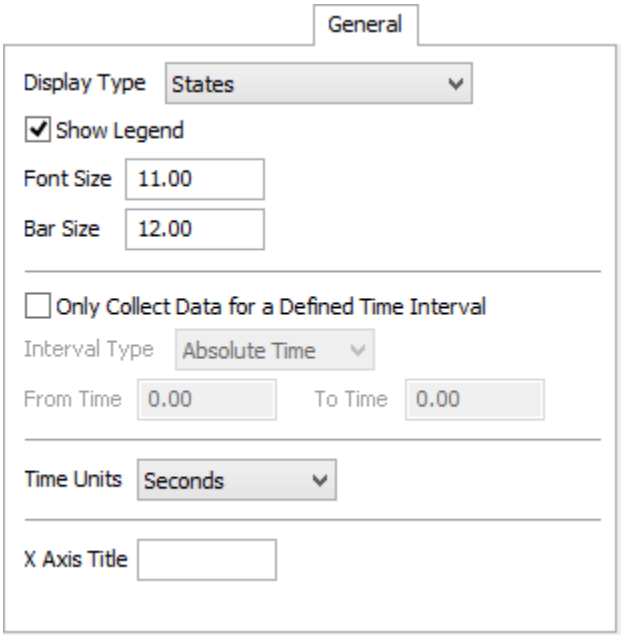
Font Size - This defines the font size of graph text.

Only Collect Data for a Defined Time Interval - This option creates a time period during which statistics for this graph will be recorded.

From Time - This defines when statistics will start being recorded.

To Time - This defines when statistics will no longer be recorded.

Gantt Chart



Display Type - This option changes the display type of graph. This may either be States or Item Trace.

Show Legend - This option adds a legend to the graph.

Font Size - This defines the font size of graph text.

Bar Size - This defines the bar height.

Only Collect Data for a Defined Time Interval - This option creates a time period during which statistics for this graph will be recorded.

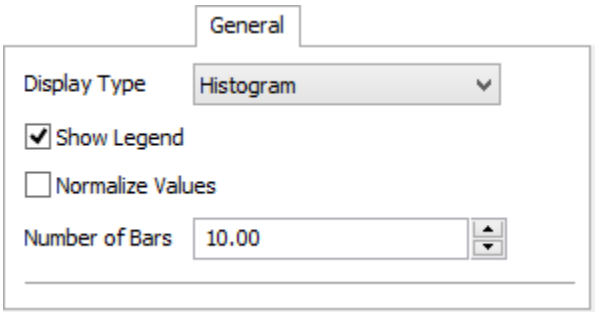
Interval Type - Some charts may have an additional option of specifying the Interval Type, like the Gantt Chart. The Interval Type can be set to:

- Absolute Time: Starts collecting data at the From Time and ends collecting data at the To Time.
- Time Window: Collects data for a specified length of time. The Length field defines the length of time from the first recorded statistic to the last recorded statistic and dynamically updates as the model runs. As the data is gathered in a model run, data from the start will be removed in order to keep all of the data in the chart within the specified length of time.

From Time - This defines when statistics will start being recorded.

To Time - This defines when statistics will no longer be recorded.

Tracked Variable Graphs



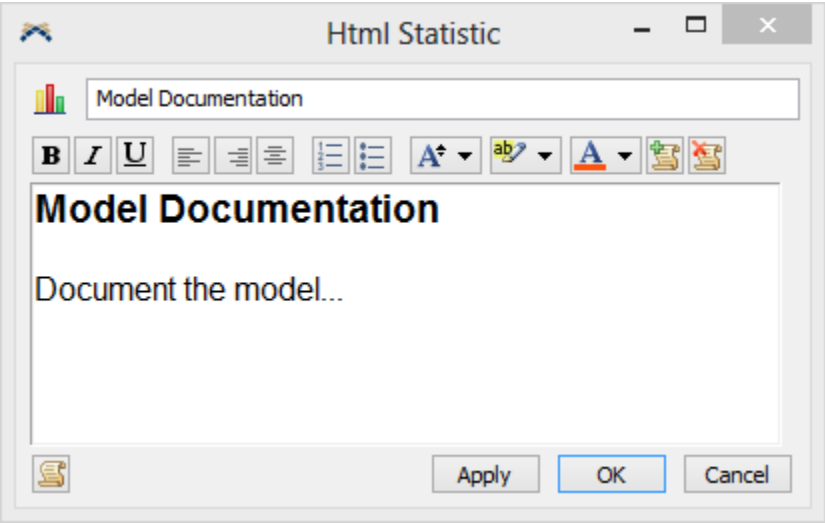
Display Type - This option changes the display style of graph.

Show Legend - This option adds a legend to the graph.

Normalize Values - This option shows the values of the histogram as a percent, rather than an actual value.

Number of Bars - This option adjust the number of bars that the histogram has.

HTML Statistic (Model Documentation)



Name Field - This field sets the name of the the documentation display graph.

- This button makes the selected text bold. If the whole selection is already bold, this button removes the bold effect.

- This button italicizes the selected text. I f the whole selection is already italicized, this button removes the italics.

- This button underlines the selected text. If the whole selection is already underlined, this button removes the underline.

- This button makes the text on the current line left justified. If multiple lines are selected, all lines are left justified.

- This button makes the text on the current line right justified. If multiple lines are selected, all lines are right justified.

- This button makes the text on the current line right justified. If multiple lines are selected, all lines are right justified

- This button places the current line in an ordered list. If multiple lines are selected, all lines are placed in the list.

- This button places the current line in an unordered list. If multiple lines are selected, all lines are placed in the list.

- This button changes the font of the current selection to be the specified size.

- This button highlights the selected text in the specified color.

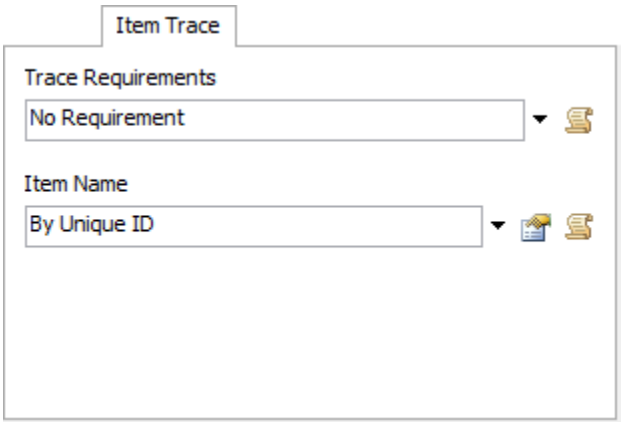
- This button changes the font color of the selected text to the specified color.

- This button inserts a section of flexscript code, which can be used to dynamically update the model documentation based on the model itself.

- This button removes the current section of flexscript.

- This button toggles the view between the visual editor and the html editor.

Item Trace Page



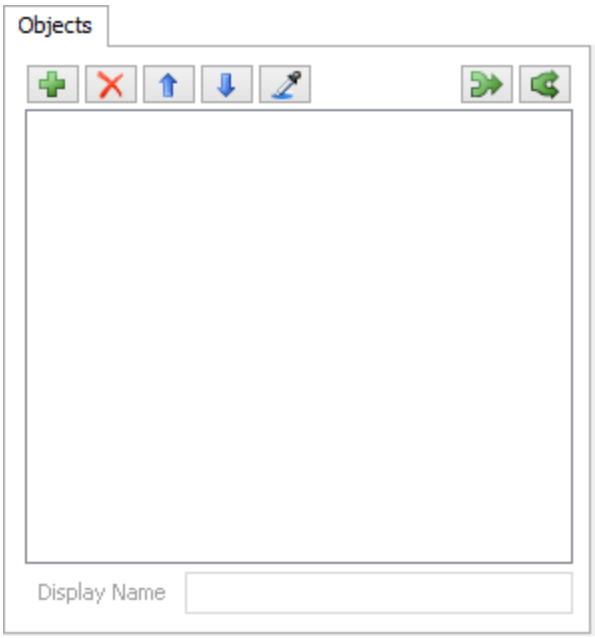
Trace Requirements - This picklist allows you to define which flowitems will be traced.

- current: The object as defined in the Objects page.
- item: The involved flowitem.

Item Name - This picklist allows you to define a custom display name for traced items.

- current: The object as defined in the Objects page.
- item: The involved flowitem.
- itemnode: The node associated with the item and its graph data. Setting the name of this node will set the display name in the graph.

Objects Page



- Adds objects to the current list of objects

- Removes objects from the current list of objects

- Moves the selected object up in the list

- Moves the selected object down in the list

- Sample an object in the model.

- Combines several objects into one object

- Splits a combined object into its components.

Display Name - This defines the name of the object or group that will be displayed in the graph. Leaving this field blank will cause the object or group name to be displayed.

State Profile - State Statistics Only. Defines which State Profile should be used. This option is only available if the first object in the object list has additional State Profiles. All objects in the list must have the selected State Profile.

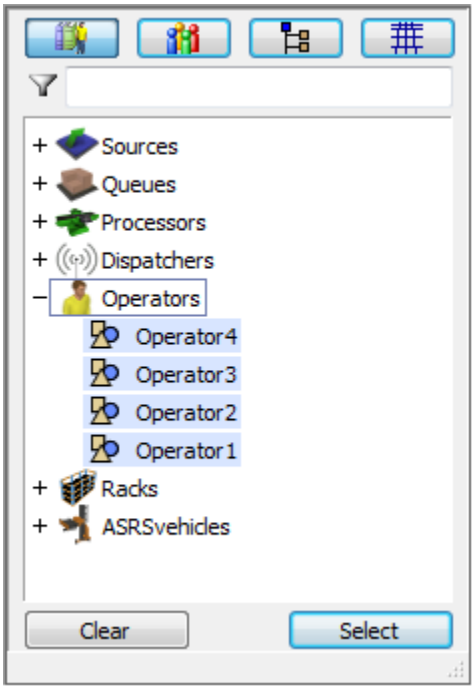
Start Trace On - Gantt Chart (Item Trace) Only. This specifies when a item trace should be started. This value can be set for each individual object.

Adding Objects

There are five ways to add objects to a graph. For more information on using the Sampler to select objects, see the Sampler page.

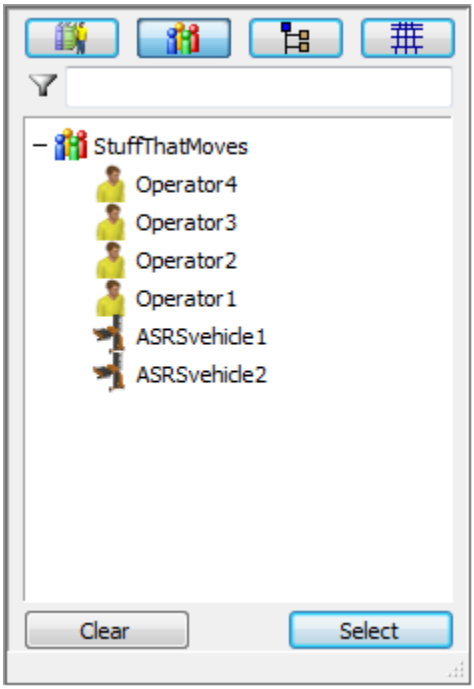
The current mode is highlighted at the top of the browsing window.

Browse by Class



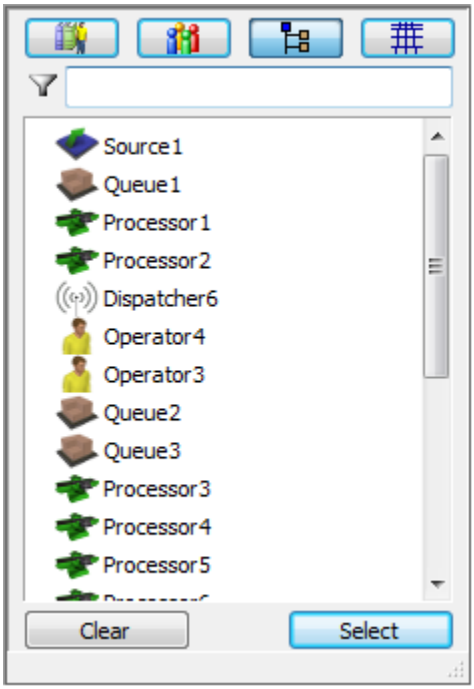
This method of adding objects sorts the objects by class. To select an entire class, click on the type icon. Click on an object to select or deselect it.

Browse by Group



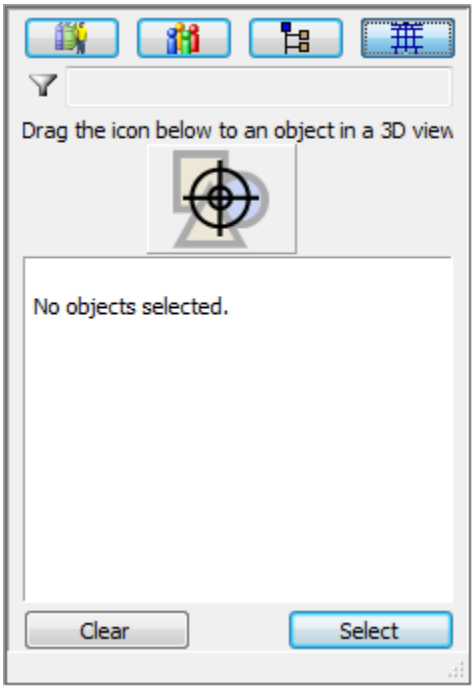
This method of adding objects sorts the objects by group. To select an entire group, click on the type icon. Click on an object to select or deselect it.

Browse by Object



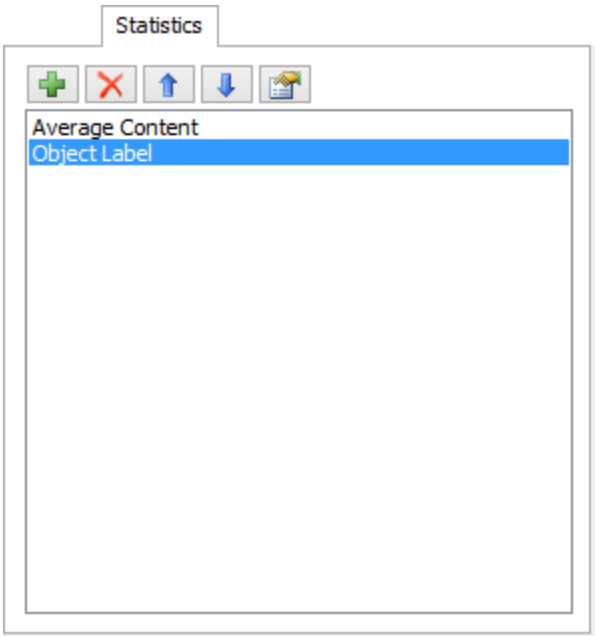
This method of adding objects lists all objects in the model. Click on an object to select or deselect it.

Select by Dragging



This method of adding objects uses the 3D model view. Simply drag the target icon from the current window to an object in the model you would like to add.

Statistics Page



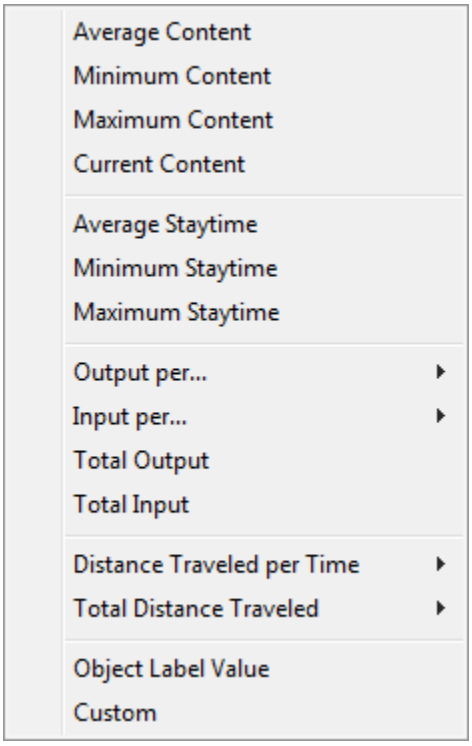
- Adds a statistic to the current list of statistics

- Removes a statistic from the current list of statistics

- Moves the selected statistic up in the list

- Moves the selected statistic down in the list

Available Statistics



Output per... - This statistic will measure output per time, where the units of time can be selected from the side list.

Input per... - This statistic will measure input per time, where the units of time can be selected from the side list.

Distance Traveled per Time - This statistic will measure the velocity of an object. Both the length units and the time units can be selected from the side lists.

Total Distance Traveled - This statistic will measure the the total distance an object travels. The length units can be selected from the side list.

Custom - This statistic allows you to execute custom code to define what value is displayed.

Tracked Variables

- Adds a tracked variable to the current list of tracked variables

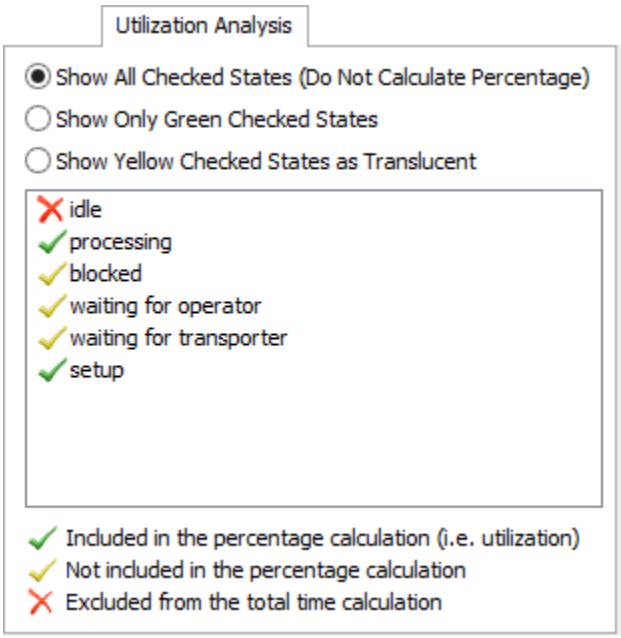
- Removes a tracked variable from the current list of tracked variables

- Moves the selected tracked variable up in the list

- Moves the selected tracked variable down in the list

For more information on tracked variables, see the Tracked Variables page.

Utilization Analysis Page



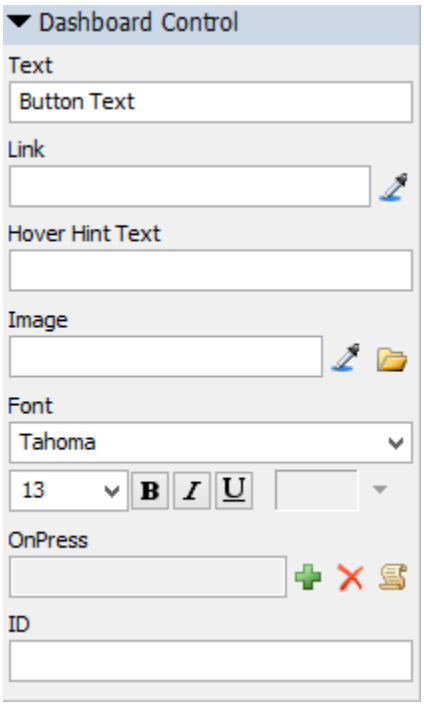
Show All Checked States - This option shows all states as solid colors. It does not calculate the percent time for which the object is utilized.

Show Only Green Checked States - This option shows the percent time that the object is in a green checked state. It does not show any other state information.

Show Yellow Checked States as Translucent - This option shows green checked states in solid green. States with yellow checks are shown in translucent colors.

State List - Valid states for the selected objects will be displayed in this list. Click on the check marks to toggle between the the three calculation options.

Model Input Properties



Text - Specify the title or text of the control.

- Available for: Static Text, Checkbox, Radio Button, Button

Link - Specify the path to a node in the model.

- Available for: Dynamic Text, Edit, Checkbox, Radio Button, Combobox, Listbox, Tracker, Table, Button

Hover Hint Text - This text will be displayed as a tooltip when the user hovers their mouse over the control.

- Available for: Dynamic Text, Edit, Checkbox, Radio Button, Combobox, Listbox, Tracker, Button

Image - Displays an image instead of text for the control.

- Available for: Static Text, Button

Font - Specifies the font name, size, properties and color of the control's text (color only available for Static Text, Dynamic Text and Edit controls).

- Available for: All

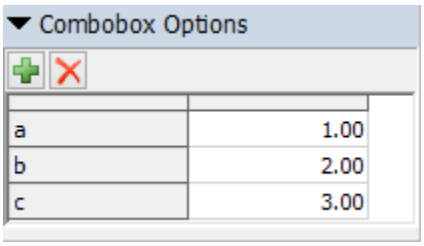
Triggers - Some model input objects have triggers that fire, allowing you to execute your own custom code.

- **OnPress -** Fires when a button, checkbox or radio button are pressed.
- **OnApply -** Fires when the enter key is pressed in an edit field and when you click off of an edit field (focus is removed).
- **OnDrag -** Fires as a tracker is clicked/dragged
- **OnSelect -** Fires when an item in the combobox/listbox is selected.

ID - A Dashboard Control's ID is a string that allows you to easily reference the control through code or picklist options. Use the getdashboardcontrol() command to get a reference to the model input object (the field, button, etc).

- Available for: All

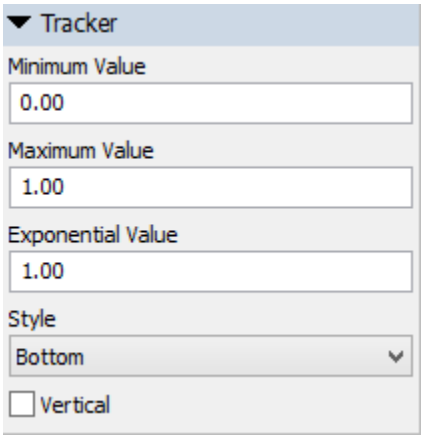
Combobox and Listbox Options



Options - A list of items to display in the combobox/listbox drop down. Values must be numeric.

Linking to string data: If a combobox or listbox is linked to a node with string data like a label, selecting an option will set the value of the linked label or node to be the name of the option, rather than its numeric value.

Tracker



Minimum Value - The minimum value of the tracker.

Maximum Value - The maximum value of the tracker.

Exponential Value - Changes the distribution of values between the minimum and maximum to be exponential. Setting this value to 1 will cause the tracker to be linear.

Style - Changes the style of the thumb button.

Vertical - If checked, the tracker will display vertically.

Reports and Statistics

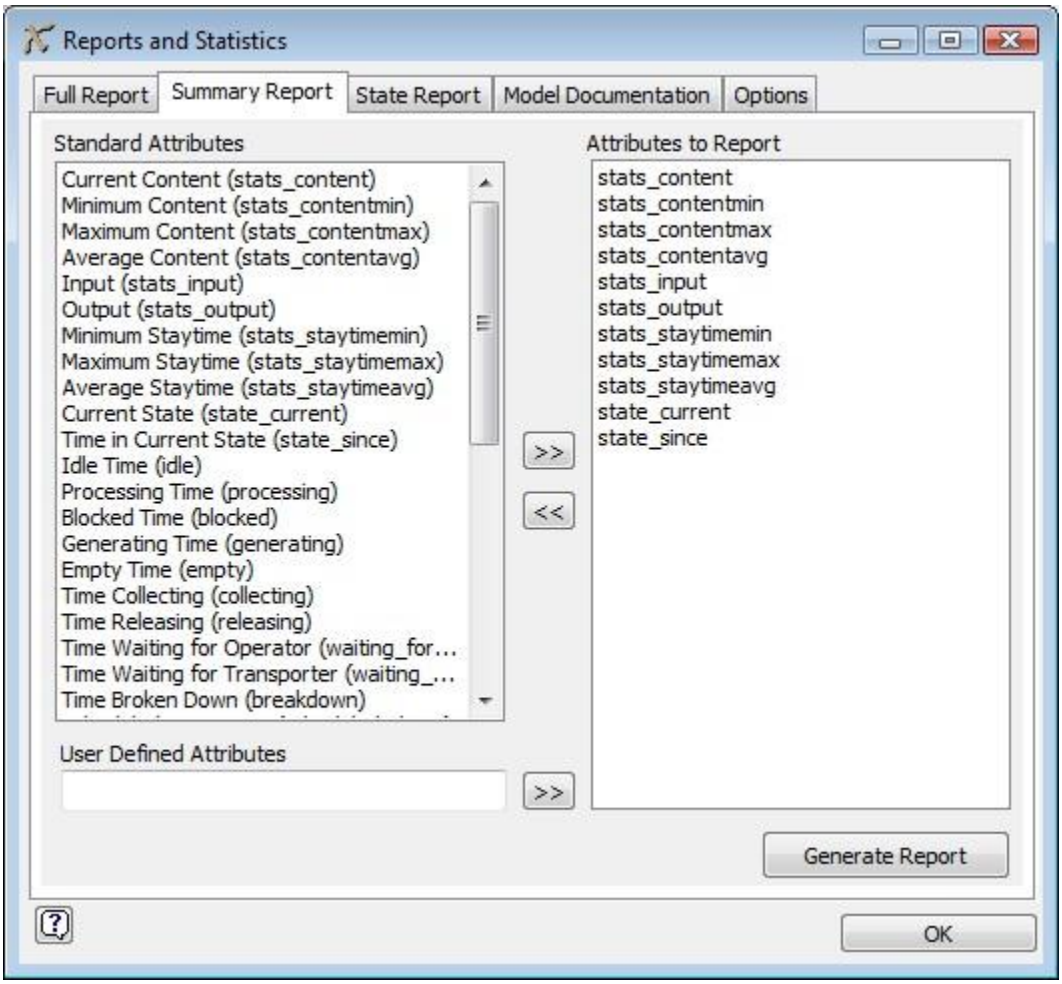
The Reports and Statistics dialog box allows the modeler to create various reports based on statistics gathered during a model run. These reports can include information about flowitem throughput, staytime, state history and other data that the modeller can select or customize. This dialog box also allows the modeler to create a document containing the most important details about objects in the model.

Reference

- **Summary Report Tab**
- **State Report Tab**
- **Model Documentation Tab**
- **Options Tab**

Summary Report Tab

The standard report tab allows you to create a report of your model. FlexSim reports on a list of attributes that you define for the model. Once you have created the list of attributes that you want to report, click on the Generate Report button. The report is then created and exported to Microsoft Excel.



Standard Attributes - This list lets you select standard attributes like content, staytime, state variable, etc. Press the >> button at the top to add the selected attribute to the list of attributes to report.

User Defined Attributes - Here you can type in the name of a label or variable that you want reported, then click the >> button at the bottom to add your own attributes to the report list. For example, if one or

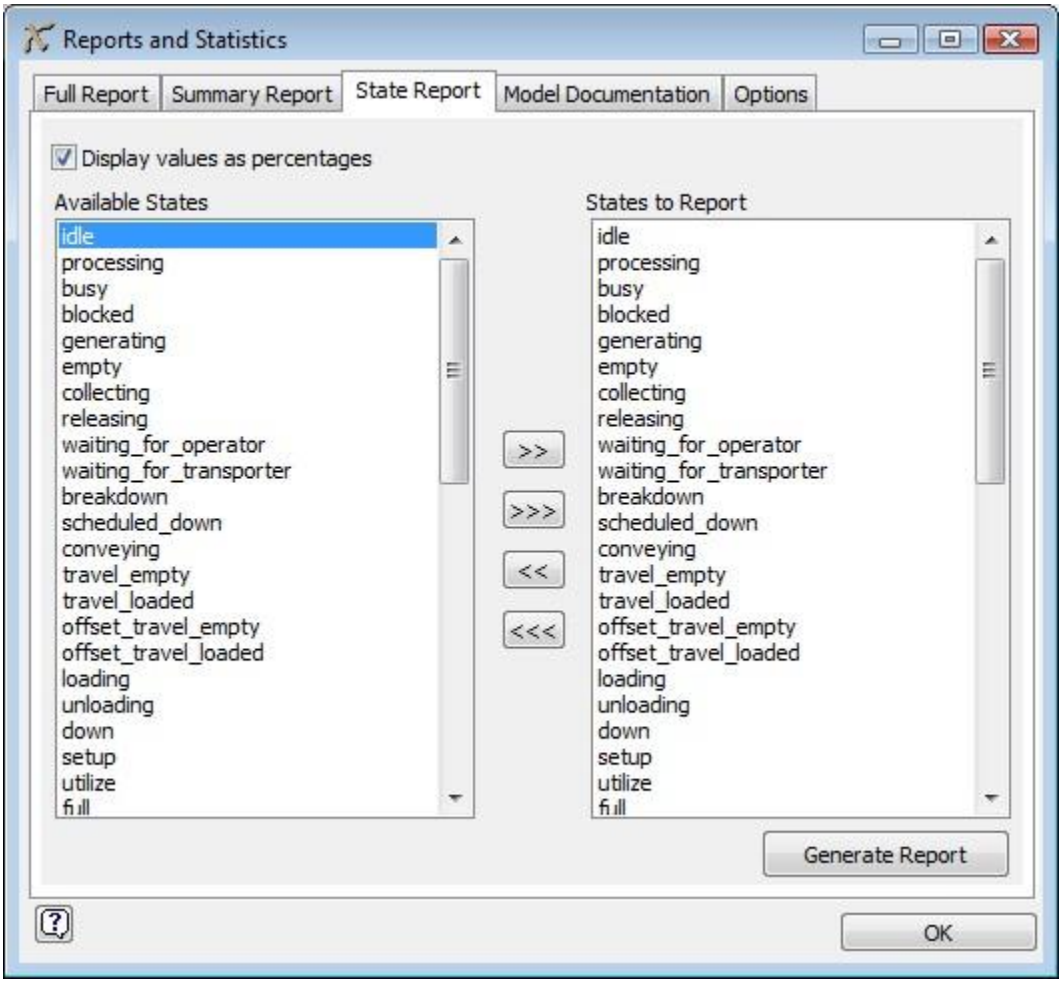
more Queues have a label called "lastreditem" and you want a report of all such labels and their values, then type "lastreditem" in the field and press the >> button.

Attributes to Report - This is the list of attributes that will be reported. To remove an item from the list, select it and press the << button.

Generate Report - Press this button to generate the report.

State Report Tab

The state report tab allows you to create a report of the time the objects in your model spent in individual states. This can be reported either as an exact time or as a percentage of the model run time. Once you have created the list of states that you want to report, click on the Generate Report button. The report is then created and exported to Microsoft Excel.



Display values as percentages - If this box is checked the report will display the percentage of the total run time that the objects spent in each state. If it is not checked, the report will display the exact amount of time the objects spent in each state.

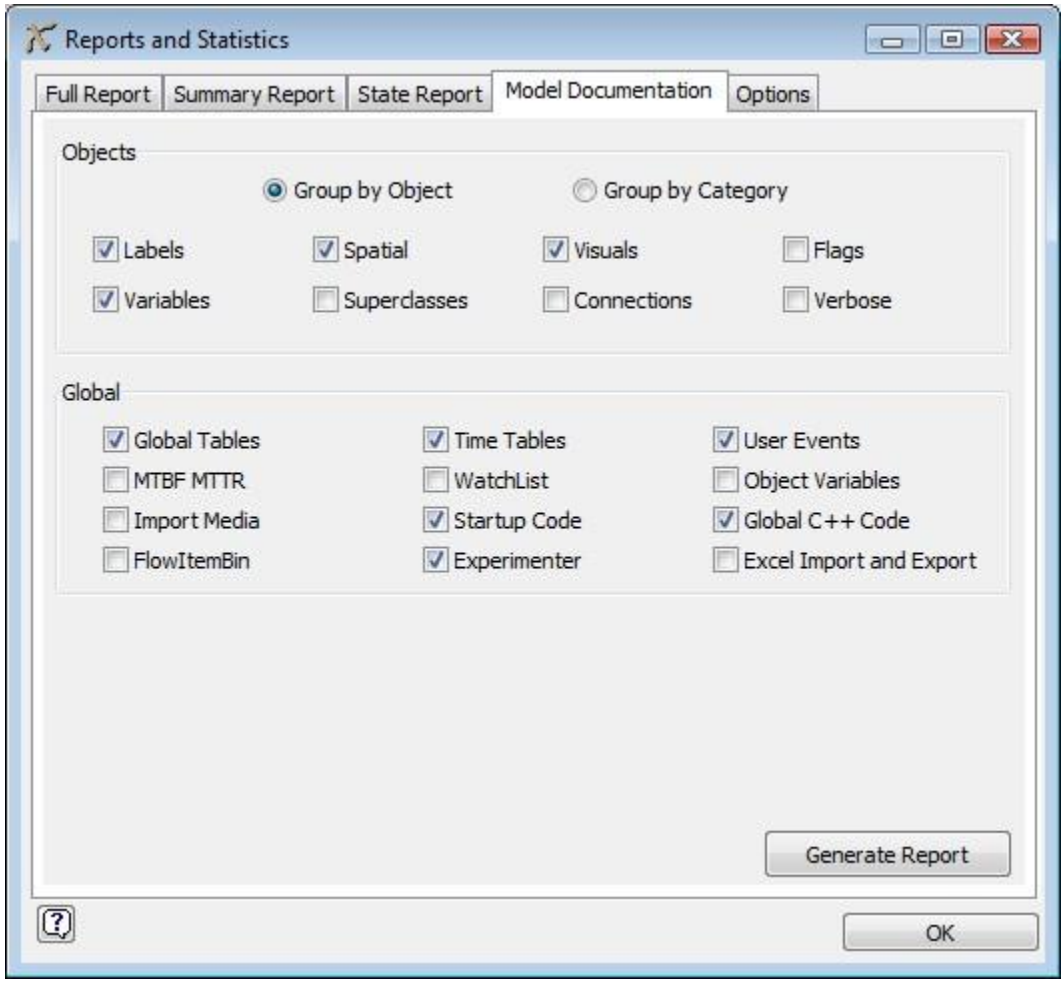
Available States - These are the states available in FlexSim that can be included in the report. To place a state in the States to Report column, highlight the state you are interested in and press the >> button. You can place all of the available states in the States to Report column by pressing the >>> button.

States to Report - The time the objects spent in these states will be displayed in the report. The time for all of these states will be reported for all objects, even if the object was never in some of the states. To remove a state from this list, highlight it and press the << button. To remove all of the states from this list press the <<< button.

Generate Report - Press this button to generate the report.

Model Documentation Tab

The model documentation tab lets you create a (.txt) document that reports information on your model. Check the appropriate boxes that you want to be documented, and then click the Generate Report button to create the file.



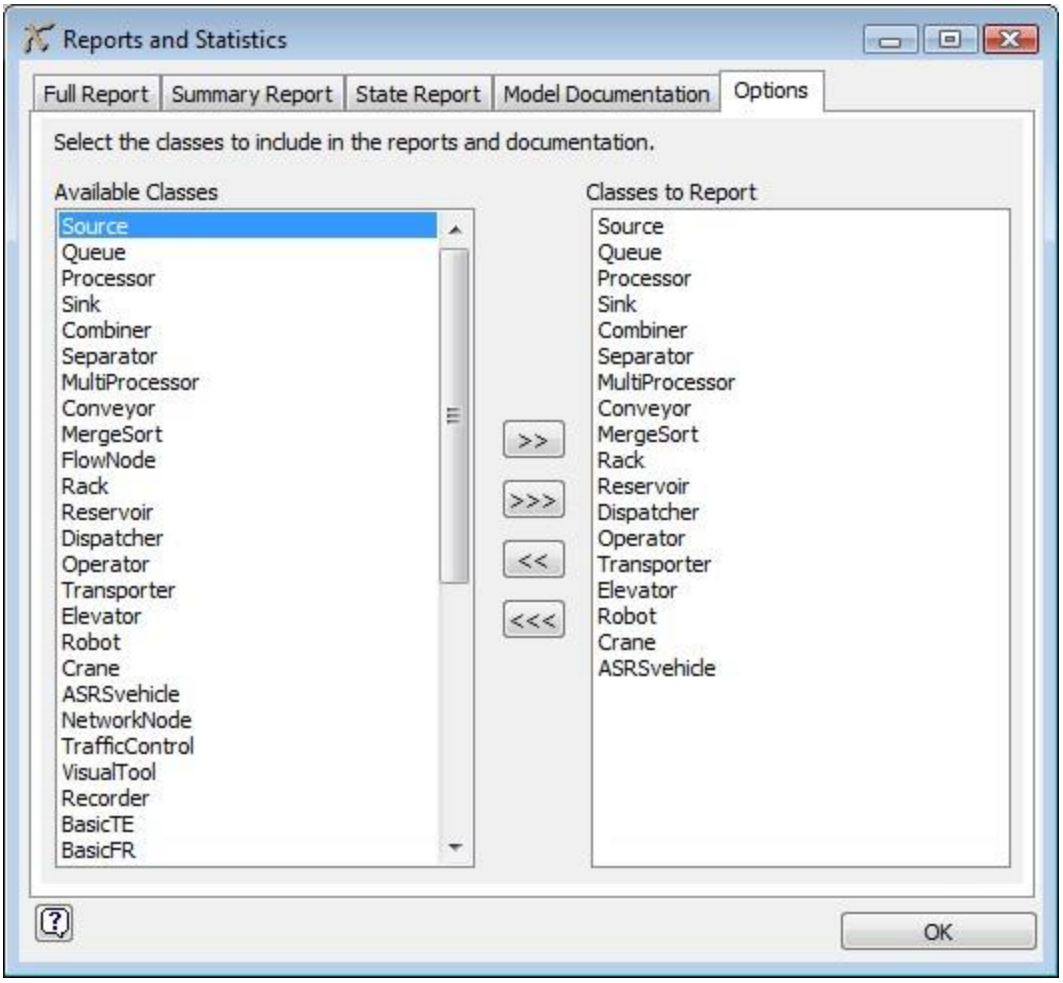
Objects - These options allow the modeler to select which attributes of the model objects should be included in the report. If "Group by Object" is selected, all of the selected attributes for each object will be together in the final report. If "Group by Category" is selected, then all of the values for each attribute will be together in the report. If "Verbose" is selected, then any code fields (triggers, process time, etc) will be documented with the full text of the field. If "Verbose" is not checked, then only the text that appears a template window will be recorded in the resulting document.

Global - These options allow the modeler to select which global objects and features they would like included in the report.

Generate Report - Press this button to generate the report.

Options Tab

The options tab allows you to select the classes of objects that will be displayed in the reports.



Available Classes - These are all of the classes in FlexSim whose instances can be included in reports. These are the classes that appear the Library Icon Grid. You can add a class to the Classes to Report list by highlighting the class you are interested in and pressing the >> button. You can add all of the classes to the Classes to Report list by pressing the >>> button.

Classes to Report - All of the objects in the model that are instances of any of these classes will be included in any reports that are generated. Any instances of classes that are not in this list will not be included in reports. To remove a class from this list, highlight the class you want to remove and press the << button. You can remove all of the classes from the list by pressing the <<< button.

FlexSim Coding

1. Writing Logic in FlexSim
2. Basic Modeling Functions
3. Code Editor
4. Debugging
5. Command Helper
6. When to Compile

Writing Logic in FlexSim

FlexSim provides two options for writing custom logic: FlexScript and C++. FlexScript is generally preferred to C++ since the code works immediately in the model without having to be compiled. When speed is an issue, C++ code runs faster than FlexScript, but must be compiled. If you want to code in FlexScript, but run with the speed of C++, you can toggle between these options in the Build menu.

FlexScript is nearly identical to C++ in its syntax and application, but is simplified for ease of use. This topic covers the programming options available in FlexScript.

Topics

- **Where to get help**
- **General Rules**
- **Variable Types**
- **Declaring and Setting Variables**
- **Math Operations**
- **Comparing Variables**
- **Relating Variables**
- **Setting and Changing Variables**
- **Executing Commands**
- **Flow Constructs**

Where to get help

Whenever you need help with what commands to use and how to use them, you can refer to the "Commands" documentation found through FlexSim's Help menu.

General Rules

Here are some general rules you will need to know when creating your own logic.

- language is case sensitive (A is not the same as a)
- no specific format is required (free use of spaces, tabs and line returns is encouraged)
- numbers are double precision floating point values unless otherwise specified.
- text strings are usually entered between quotes. “mytext"
- parenthesis follow a function call and commas separate the arguments of the function. moveobject(object1,object2);
- a function or command will always end with a semi-colon
- parenthesis can also be used freely to make associations in your math and logic statements.
- curly braces are used to define a block of statements.
- to comment out the rest of a line use //
- don’t use spaces or special characters in name definitions (_ is ok).
- named variables and explicit values can be interchanged in writing expressions.

Variable Types

FlexSim uses just four variable types. Each of the four types can also be used in an array structure. The following explains each of these types.

Single Variables

Type	Description
int	**integer type**
double	**double precision floating point type**
string	**text string**
treenode	**reference to a FlexSim node or object**

Array Variables

Type	Description
intarray	**an array of integer types**
doublearray	**an array of double types**
stringarray	**an array of string types**
treenodearray	**an array of treenode types**

For more information on how the treenode (or FlexSim node) type works, refer to the FlexSim tree structure.

Declaring and Setting Variables

The following are some examples of how to declare and set variables.

```
int index = 1;

double weight = 175.8;
string category = "groceries";
treenode nextobj = next(current);
```

Declaring and Setting Array Variables

The following are examples of how to use array types.

```
intarray indexes = makearray(5); // makes an array with 5 elements


indexes[1] = 2; // in FlexSim, arrays are 1-based
indexes[2] = 3;
indexes[3] = 2;
indexes[4] = 6;
indexes[5] = 10;
```

```
doublearray weights = makearray(3);

fillarray(weights, 3.5, 6.7, 1.4); // fillarray is a quick way of setting the array values

stringarray fruits = makearray(2);

fruits[1] = "Orange";
fruits[2] = "Watermelon";


treenodearray operators = makearray(4);

operators[1] = centerobject(current, 1);
operators[2] = centerobject(current, 2);
operators[3] = centerobject(current, 3);
operators[4] = centerobject(current, 4);
```

Math Operations

The following list show different math operations that can be performed on values.

Operation	Floating Point Example (=solution)	Integer Example (=solution)
+	**1.6+4.2 (=5.8)**	**2+3 (=5)**
-	**5.8-4.2 (=1.6)**	**5-2 (=3)**
*****	**1.2 * 2.4 (=2.88)**	**3*4 (=12)**
/	**6.0/4.0 (=1.5)**	**20/7 (=2)**
% (integer mod)		**34%7(=6)**
sqrt()	**sqrt(5.3) (=2.3)**	
pow()	**pow(3.0,2.2) (=11.2)**	**pow(3,2) (=9)**
round()	**round(5.6) (=6)**	
frac()	**frac(5.236) (=0.236)**	
fabs()	**fabs(-2.3) (=2.3)**	

fmod() (floating point mod)	**fmod(5.3,2) (=1.3)**	

Be aware as you write your logic that, by default, all values in FlexSim are double precision floating point, so you will usually be using the operations as they apply to floating point numbers.

Note: By performing operations on floating point numbers, some precision may be lost.

Note: Be careful in using these operations while mixing integer types with floating point types, or with using just integer types. For example, the / operator will return an integer if both operators are integers. This may not be what you want to get out of the operation, in which case you will need to use floating point types instead of integer types. Note also that C++ will interpret the literal number 5 as an integer type. If you want it to interpret the number as a floating point type, enter 5.0 instead of just 5.

Comparing Variables

The following table shows different operations for comparing two values or variables.

Operation	Example (solution)
> (greater than)	**1.7>1.7 (false)**
< (less than)	**-1.7 < 1.5 (true)**
>= (greater than or equal to)	**45 >= 45 (true)**
<= (less than or equal to)	**45 <= 32 (false)**
== (equal to)	**45 == 45 (true)**
!= (not equal to)	**45 != 35 (true)**
comparetext()	**comparetext(getname(current),"Processor5")**

Warning: The == operator can often cause problems if you are comparing two double precision floating point values, and one or both of those values have been calculated using math operations. When doing math operations, floating point values may lose some precision. Since the == operator will only return true if all 64 bits of each value are exactly the same, even a small precision loss will cause the == operator to return false. In such cases, you will want to instead verify that the two values are within a range of each other. For example: fabs(value1 - value2) < 0.000001, will return true if the two values are practically equal for all intents and purposes.

Relating Variables

The following table shows different operations for relating several comparisons.

Operation	Example

&& (logical AND)	**x>5 && y<10**
|| (logical OR)	**x==32 || y>45**
! (logical NOT)	**!(x==32 || y>45)**
min()	**min(x, y)**
max()	**max(x, y)**

Setting and Changing Variables

The following tables show ways of setting and changing variables.

Operation	Example
=	**x = x + 2;**
+=	**x += 2; (same as x = x + 2)**
-=	**x -= 2; (same as x = x - 2)**
***=**	**x *= 2; (same as x = x * 2)**
/=	**x /= 2; (same as x = x / 2)**
++	**x ++; (same as x = x + 1)**
--	**x --; (same as x = x - 1)**

Executing Commands

Executing a command in FlexSim is made of following parts. First type command's name, followed by an open parenthesis. Then enter each parameter of the command, separating multiple parameters by commas. Each parameter can be a variable, an expression of variables, or even a command itself. Finish the command with a close parenthesis, and a semi-colon. For detailed information on the commands, their functionality and parameter lists, refer to the "Commands" documentation found through FlexSim's Help menu. For a quick reference of the most used commands in FlexSim, refer to the section on basic modeling functions.

Syntax	Examples

`commandname(parameter1,parameter2,parameter3...);`	`coloryellow(current);`
	
	`setrank(item, 3 + 7);`

	`setitemtype(item, getlabelnum(current, "curitemtype"));`

Flow Constructs

The following are constructs which allow you to modify the flow of your code.

Logical If Statement

The if statement allows you to execute one piece of code if an expression is true, and another piece of code if it is false. The else portion of the construct is optional.

Construct	Examples

`if (test expression)`
`{`
`code block`
`}`
`else`
`{`
`code block`
`}`
	`if (content(item) == 2)`
	`{`
	`colorred(item);`
	`}`
	`else`
	`{`
	`colorblack(item);`
	`}`

Logical While Loop

The while loop will continue to loop through its code block until the test expression becomes false.

Construct	Examples

`while (test expression)`
`{`
`code block`
`}`
	`while (content(current) == 2)`
	`{`
	`destroyobject(last(current));`
	`}`

Logical For Loop

The for loop is like the while loop, except that it is used usually when you know exactly how many times to loop through the code block. The start expression is executed only once, to initialize the loop. The test expression is executed at the beginning of each loop and the loop will stop as soon as this expression is false, just like the while loop. The count expression is executed at the end of each loop, and typically increments some variable, signifying the end of one iteration.

Construct	Examples

`for(start expression;test expression;count expression)`
`{`
`code block`
	`for (int index = 1;index <= content(current);index++)`
	`{`
	`colorblue(rank(current,index));`

}	}

Logical Switch Statement

The switch statement allows you to choose one piece of code to execute out of several possibilities, depending on a variable to switch on. The switch variable must be an integer type. The example below sets the color of items of type 1 to yellow, type 5 to red, and all other types to green.

Construct	Examples

switch (*switchvariable*)
{
  case *casenum*:
  {
  code block;
  break;
  }
  default:
  {
  code block;
  break;
  }
}
	int type = getitemtype(item);
	switch (type)
	{
	  case 1:
	  {
	  coloryellow(item);
	  break;
	  }
	  case 5:
	  {
	  colorred(item);
	  break;
	  }
	  default:
	  {
	  colorgreen(item);
	  break;
	  }
	}

Redirection

Each of the flow constructs described can be redirected mid-execution with either a continue, break or return statement. The following explains how each of these statements work.

Construct	Examples
continue;	Only valid in For and While loops. Halts the current iteration of the loop and goes on to the next iteration in the loop. In a For loop the counter is incremented before continuing.
break;	Only valid in For, While and Switch statements. Breaks out of the current For, While or Switch block and continues with the line immediately following this block. Nested statements only break out of the current statement and continue on in the containing statement.
return 0;	Returns out of the current method entirely and continues with the line following the code that called this method. The 0 is required in Flexscript commands (picklists and

	triggers included) because all Flexscript commands return a double type.

Basic Modeling Functions and Logic Statements

Here we have provided a quick reference for commands that are used most often in FlexSim. For more detailed information on these commands, refer to the command index.

Topics

- **Referencing Commands**
- **Object Attributes**
- **Object Spatial Attributes**
- **Object Statistics**
- **Object Labels**
- **Object Control**
- **Object Variables**
- **Tables**
- **TaskExecuter Control**
- **Prompts and Printouts**
- **Advanced Functions**

Object Referencing

The following commands and access variables are used in referencing objects in FlexSim.

current and item

- **current -** the current variable is a reference to the current resource object. It is often an access variable in pick lists.
- **item -** the item variable is a reference to the involved item for a trigger or function. It is often an access variable in pick lists.

Referencing Commands

command(parameter list)	Explanation	Example
first(node)	This returns a reference to the first ranked object inside of the object passed	**first(current)**
last(node)	This returns a reference to last ranked object inside of the object passed	**last(current)**
rank(node, ranknum)	This returns a reference to the object at a given rank inside the object passed	**rank(current, 3)**
inobject(object, portnum)	This returns a reference to the object connected to the input port number	**inobject(current, 1)**

	of the object passed	
outobject(object, portnum)	This returns a reference to the object connected to the output port number of the object passed	**outobject(current, 1)**
centerobject(object, portnum)	This returns a reference to the object connected to the center port number of the object passed	**centerobject(current, 1)**
next(node)	This returns a reference to the next ranked object of the object passed	**next(item)**
prev(node)	This returns a reference to the previous ranked object of the object passed	**prev(item)**
node(path, startnode)	This returns the object found at path in the tree beginning from startnode	**node("/Floor1/Processor", model())**

Object Attributes

command(parameter list)	Explanation
getname(object)	This returns the name of the object
setname(object, name)	This sets the name of the object
getitemtype(object)	This returns the itemtype value of the object
setitemtype(object, num)	This sets the itemtype value of the object
setcolor(object, red, green, blue)	This sets the color of the object
colorred(object) **colorgreen, blue, white, random...**	This sets the color of the object to red, blue, green, white, etc.
setobjectshapeindex(object, indexnum)	This sets the 3D shape of the object
setobjecttextureindex(object, indexnum)	This sets the 3D texture of the object

Object Spatial Attributes

command(parameter list)	Explanation
xloc(object) **yloc(object)** **zloc(object)**	These commands return the x, y, and z locations of the object
setloc(object, xnum, ynum, znum)	This sets the x, y, and z location of the object
xsize(object) **ysize(object)** **zsize(object)**	These commands return the x, y, and z size of the object
setsize(object, xnum, ynum, znum)	This sets the x, y, and z size of the object
xrot(object) **yrot(object)** **zrot(object)**	These commands return the x, y, and z rotation of the object
setrot(object, xdeg, ydeg, zdeg)	This sets the x, y, and z rotation of the object

Object Statistics

command(parameter list)	Explanation
content(object)	This returns the current content of the object
getinput(object)	This returns the input statistic of the object
getoutput(object)	This returns the output statistic of the object
setstate(object, statenum)	This sets the current state of the object.
getstatenum(object)	This returns the current state value of the object
getstatestr(object)	This returns the current state of the object as a string
getrank(object)	This returns the rank of the object
setrank(object, ranknum)	This sets the rank of the object

getentrytime(item)	This returns the time the flow item entered the object it is currently in
getcreationtime(item)	This returns the time the flow item was created

Object Labels

command(parameter list)	Explanation
getlabelnum(object, "labelname") **getlabelnum(object, labelrank)**	This returns the value of the object's label
setlabelnum(object, "labelname", value) **setlabelnum(object, labelrank, value)**	This sets the value of the object's label
getlabelstr(object, "labelname")	This gets the string value of the object's label
setlabelstr(object, "labelname", value) **setlabelstr(object, labelrank, value)**	This sets the string value of the object's label
label(object, "labelname") **label(object, labelrank)**	This returns a reference to the label as a node. This command is often used if you have a label that is used as a table.
assertlabel(object, "labelname", DATATYPE_NUMBER) **assertlabel(object, "labelname", DATATYPE_STRING)**	This command creates a label called "labelname" if it doesn't currently exist on object, or returns a reference to the label if it already exists.

Object Control

command(parameter list)	Explanation
closeinput(object)	This closes the input of the object
openinput(object)	This re-opens the input of the object
closeoutput(object)	This closes the output of the object
openoutput(object)	This re-opens the output of the object

sendmessage(toobject, fromobject, parameter1, parameter2, parameter3)	This causes the message trigger of the object to fire
senddelayedmessage(toobject, delaytime, fromobject, parameter1, parameter2, parameter3)	This causes the message trigger of the object to fire after a certain delay time
stopobject(object, downstate)	This tells the object to stop whatever its operation is and go into the given state
resumeobject(object)	This allows the object to resume whatever its operation is
stopoutput(object)	This closes the output of the object, and accumulates stopoutput requests
resumeoutput(object)	This opens the output of the object once all stopoutput requests have been resumed
stopinput(object)	This closes the input of the object, and accumulates stopinput requests
resumeinput(object)	This opens the input of the object once all stopinput requests have been resumed
insertcopy(originalobject, containerobject)	This inserts a new copy of the object into the container
moveobject(object, containerobject)	This moves the object out of its current container into its new container

Object Variables

command(parameter list)	Explanation
getvarnum(object, “variablename”)	This returns the number value of the variable with the given name
setvarnum(object, “variablename”, value)	This sets the number value of the variable with the given name
getvarstr(object, “variablename”)	This returns the string value of the variable with the given name
setvarstr(object, “variablename”,	This sets the string value of the variable with the given

string)	name
getvarnode(object, “variablename”)	This returns a reference to the variable with the given name as a node

Other

Tables

command(parameter list)	Explanation
gettablenum("tablename", rownum, colnum) **gettablenum(tablenode, rownum, colnum)** **gettablenum(tablerank, rownum, colnum)**	This returns the value in the specified row and column of the table
settablenum("tablename", rownum, colnum, value) **settablenum(tablenode, rownum, colnum, value)** **settablenum(tablerank, rownum, colnum, value)**	This sets the value in the specified row and column of the table
gettablestr("tablename", rownum, colnum) **gettablestr(tablenode, rownum, colnum)** **gettablestr(tablerank, rownum, colnum)**	This returns the string value in the specified row and column of the table
settablestr("tablename", rownum, colnum, "value") **settablestr(tablenode, rownum, colnum, "value")** **settablestr(tablerank, rownum, colnum, "value")**	This sets the string value in the specified row and column of the table
settablesize("tablename", rows, columns) **settablesize(tablenode, rows, columns)** **settablesize(tablerank, rows, columns)**	This sets the size of the table in rows and columns
gettablerows("tablename") **gettablerows(tablenode)** **gettablerows(tablerank)**	This returns the number of rows in the table
gettablecols("tablename") **gettablecols(tablenode)** **gettablecols(tablerank)**	This returns the number of columns in the table
clearglobaltable("tablename")	Sets all number values in the table to 0

clearglobaltable(tablenode) **clearglobaltable(tablerank)**	

TaskExecuter Control

For more information on controlling TaskExecuters, refer to the task sequence section.

Prompts and Printouts

command(parameter list)	Explanation
pt(text string)	Prints text to the output console
pf(float value)	Prints a floating point value to the output console
pd(discrete value)	Prints an integer value to the output console
pr()	Creates a new line in the output console
msg("title", "caption")	Opens a simple Yes, No, Cancel dialog
userinput(targetnode, "prompt")	Opens a dialog box where you can set the value of a node in the model
concat(string1, string2, etc.)	This returns the string concatenation of two or more strings

Advanced Functions

Here are more advanced functions that you might use. We do not provide their parameter lists here. For more information, refer to the command summary.

Node commands - node(), nodeadddata(), getdatatype(), nodetopath(), nodeinsertinto(), nodeinsertafter(), getnodename(), setnodename(), getnodenum(), getnodestr(), setnodenum(), setnodestr(), inc()

Data changing commands - stringtonum(), numtostring(), tonum(), tonode(), apchar()

Node table commands - setsize(), cellrc(), nrows(), ncols()

Node table commands - setsize(), cellrc(), nrows(), ncols()

Model run commands - cmdcompile(), resetmodel(), go(), stop()

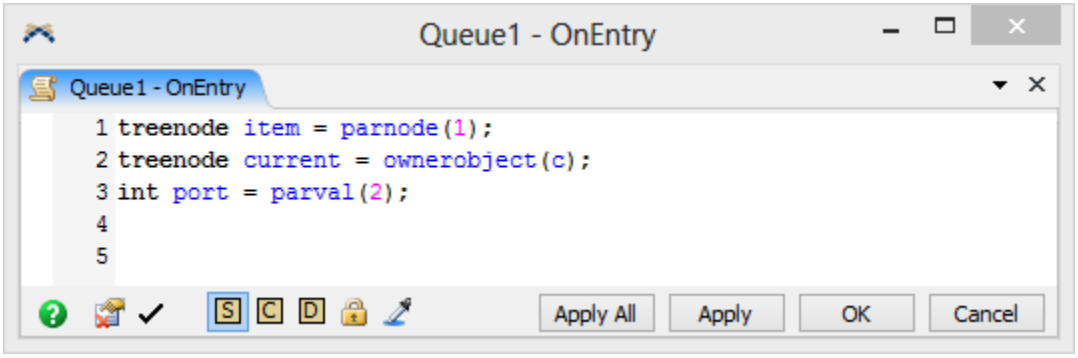
3D custom draw code commands - drawtomodelscale(), drawtoobjectscale(), drawsphere(), drawcube(), drawcylinder(), drawcolumn(), drawdisk(), drawobject(), drawtext(), drawrectangle(), drawline(), spacerotate(), spacetranslate(), spacescale()

Excel commands - excellaunch(), excelopen(), excelsetsheet(), excelreadnum(), excelreadstr(), excelwritenum(), excelwritestr(), excelimportnode(), excelimporttable(), excelclose(), excelquit()

ODBC commands - dbopen(), dbclose(), dbsqlquery(), dbchangetable(), dbgetmetrics(), dbgetfieldname(), dbgetnumrows(), dbgetnumcols(), dbgettablecell(), dbsettablecell()

Kinematics - initkinematics(), addkinematic(), getkinematics(), updatekinematics(), printkinematics()

Code Editor



The Code Editor window allows you to edit code for picklists and triggers throughout FlexSim. The window can be used as a floating window (default), or it can be docked into the main FlexSim window in any configuration. Just click the tab or the title bar and drag it over the Dock Windows icon.

Throughout FlexSim you will see icons. Clicking this icon will open the Code Edit window and allow you to edit that picklist or trigger's code. Alternatively, many right click menu's (like in the Tree Window) have the option **Explore > As Code** that will also open a Code Edit window.

When you open up the Code Editor, you'll likely see some header statements that will look something like this:

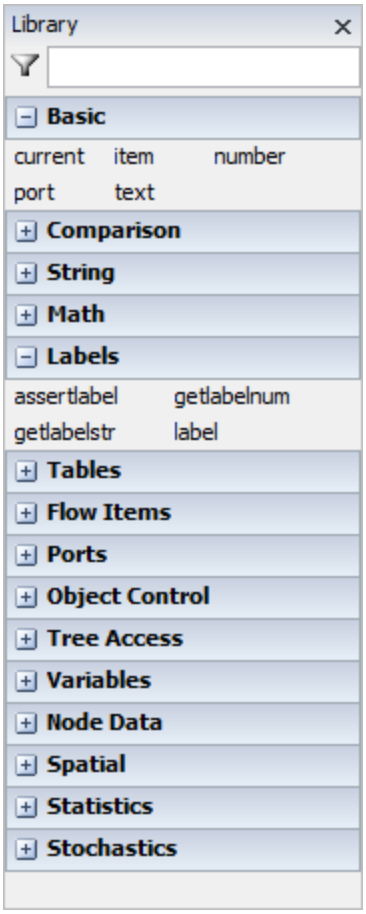
```
treenode current = ownerobject(c);
treenode item = parnode(1);
```

For more information about item and current see the Item and Current section.

Within the code window, you can specify whether you want your code to be interpreted as FlexScript or compiled as C++ (in which case you will need to compile your model). You can also check the FlexScript syntax by pressing the ✔ button.

Code Builders

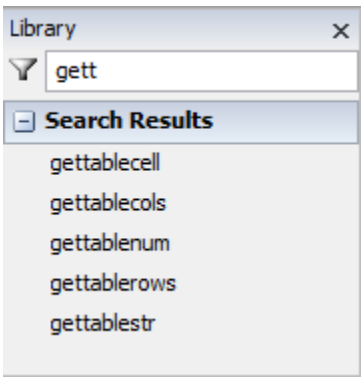
When you are editing code in the Code Editor, or entering values in a picklist field, the Library Icon Grid changes to display a list of Code Builders.



These Code Builders can be dragged into your Code Editor or picklist field to give you the correctly formatted command. Use the tab key to select each commented section of the template code. ie /*"labelname"*/

```
1 treenode item = parnode(1);
2 treenode current = ownerobject(c);
3
4 setlabelstr(current, /*"labelname"*/, /*value*/);
```

You can also filter the Code Builder list by typing in the field.

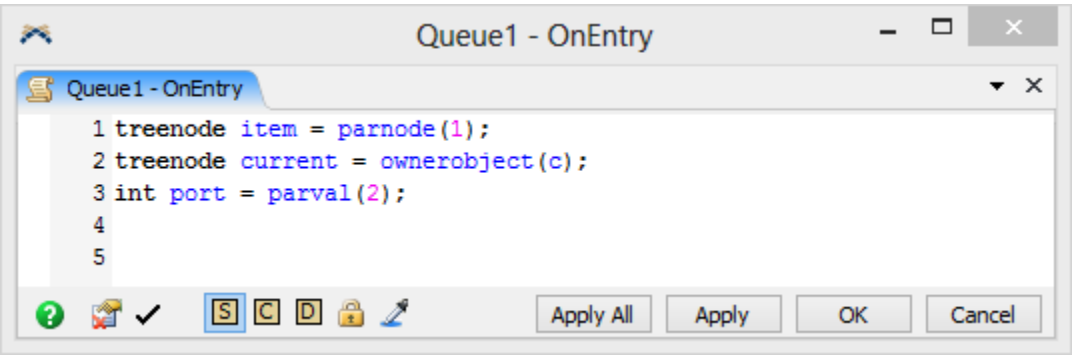


DLL Functionality

You can also specify the given field as accessing a function from a dll. In this case you would not provide the code as the text, but would provide the path to the dll as well as the name of the function to call. To create such dll you would need to use a special Visual C++ project. This project is available on the user community. The code field itself will need to specify two strings, each enclosed in quotes. The first string is the path to the dll. The second string is the name of the function. When you press the DLL radio button a message will appear that will let you create a template specifying the two strings.

Locking Code

There is also a "Locked" checkbox at the bottom of the view. This checkbox should only apply to FlexScript or C++ code. It lets you lock the code state of the field to either FlexScript or C++. In the main Build menu, there are two options to make all code either C++ or FlexScript. We provide this option so that modellers can have both the ease of use of FlexScript (code works immediately when editing in FlexScript, without having to compile) as well as the run-speed of C++ (since it is compiled, it runs much faster than FlexScript). While in the model building phase you can use FlexScript, so that your code is interpreted immediately after you write it. Then, once your model is ready to run, you can choose the Build>Make all code C++ option, compile, and run to get the speed of C++. However, there may be some code that you write that cannot be converted from FlexScript to C++ or vice versa because it uses features specific to that language. In this case you would click the to lock the code state of the given field. This would also be important if you chose one of the menu options: Make all code C++ or Make all code FlexScript.



Tab Bar (Queue1 - OnEntry) - Displays the current object and trigger/picklist being edited.

- Displays this help page.

- Removes all template code. Template code may be found in picklists and triggers and takes the form of: /***popup:Conditional*/ and /** \nCondition: *//***tag:expression*//**/

- Checks syntax for compile errors.

S - Toggles the current code as Flexscript.

C - Toggles the current code as C++ code. Editing C++ code requires compiling the model. See When to Compile.

D - Toggles the current code as DLL linked.

- Locks the toggled state of the code. This does not lock the code from being edited, but rather locks the Flexscript, C++ and DLL toggle. This can be necessary if you want your object triggers to be C++ code as the property editors automatically toggle the code to be Flexscript.

- The Sampler allows you to insert code into your code editor to reference objects, set labels, get values etc. For more information see the Sampler page.

Apply All - Saves all changes to all code editors currently docked in the same window.

Apply - Saves changes to the currently active code.

Ok - Saves changes and closes the currently active code.

Cancel - Cancels any unsaved changes and closes the currently active code.

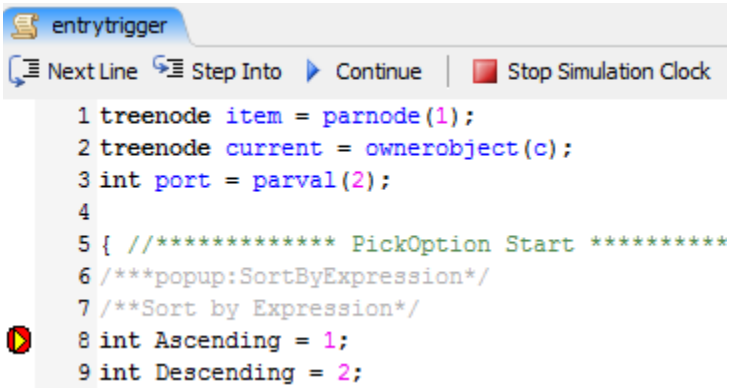
Debugging

1. Overview
2. Breakpoints
3. Call Stack
4. Code Profiler
5. Event List
6. Event Log
7. Local Variables
8. Watch Variables

Debugging Overview

How It Works

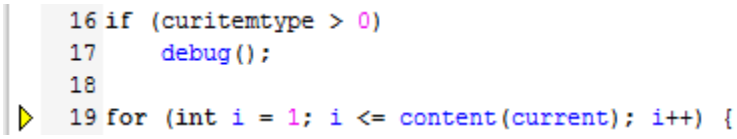
Within the FlexSim code editor, there is a margin on the left side of the line numbers. By clicking in the margin, you can add a breakpoint to that line of code. The breakpoint will appear as a in the margin. You can delete the breakpoint by clicking the . When a line of code with a breakpoint is executed, FlexSim will enter debug mode. While debugging, you will only be able to interact with certain areas of the program including the debugger tools the tree window, output consoles and a limited number of other windows. The code window will change to give you tools for debugging.



The debug() Command

The `debug()` command can be placed in any flexscript code. It acts just like a breakpoint; it will pause the model and open the debug window when it is executed. However, it will pause the model even when breakpoint debugging is disabled.

This command makes it possible to have conditional breakpoints.



Controls

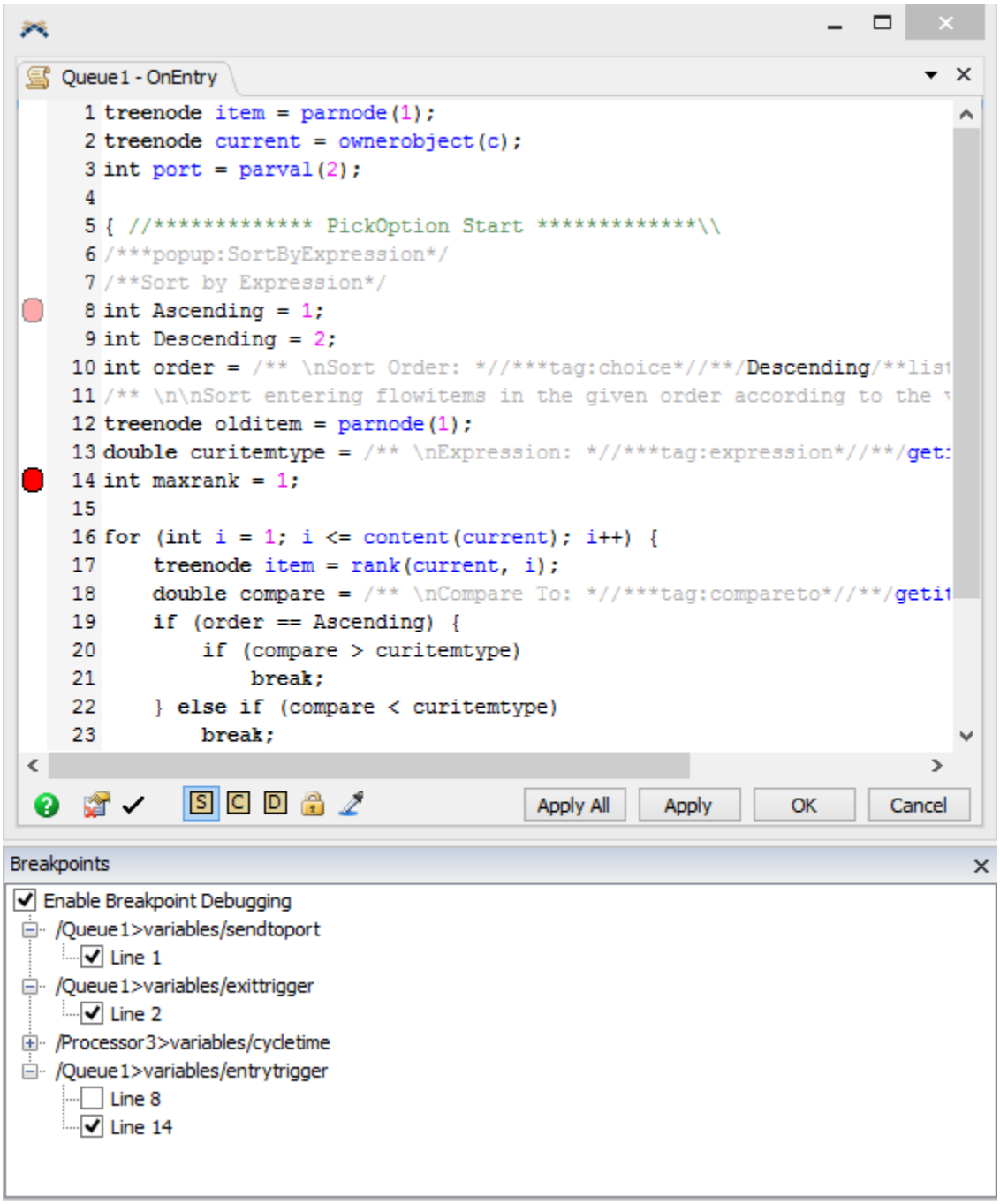
Next Line - The yellow arrow shows you what line will be executed next. By pushing the Next Line button, that line will be executed and the yellow arrow will move to the next line of code to be executed.

Step Into - This button can be used when a line of code contains certain function calls. It allows you to follow the code execution into the function and proceed line by line. When the function is finished, the debugger will return to the code that called the function. The only functions you can follow in this way are **nodefunction**, **executefsnode**, and any custom **user commands**.

Continue - This will cause the FlexScript execution to continue until it reaches another breakpoint. If the code currently being executed finishes, then it will leave debug mode.

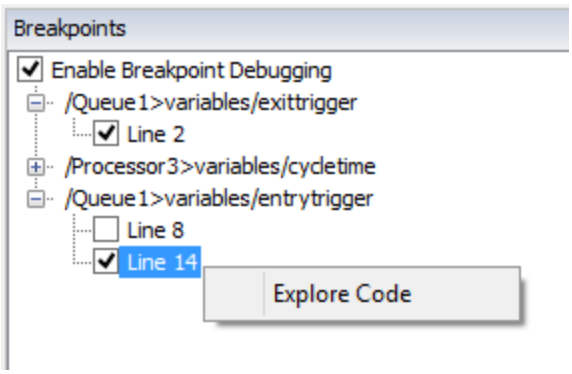
Stop Simulation Clock - This button will stop the model's simulation clock. This is particularly useful if the model is running quickly, as the **Continue** button may cause FlexSim to re-enter debug mode almost instantly when it continues.

Breakpoints

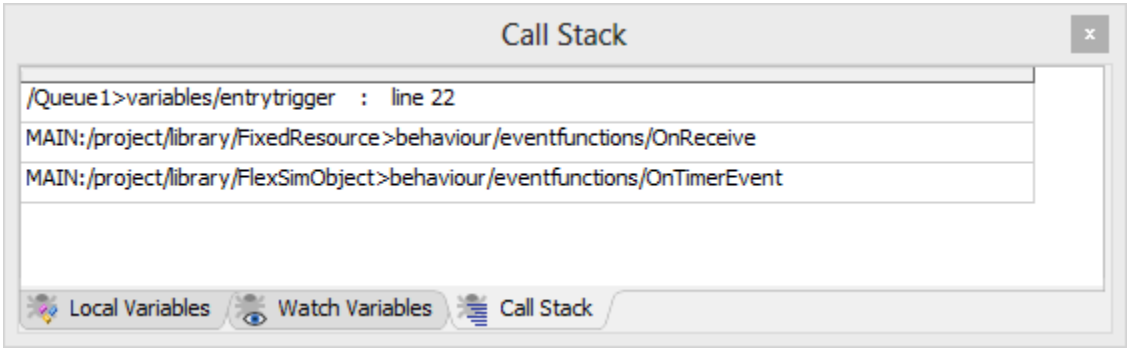


The Breakpoints window is available through the Debug menu. It is a treeview with checkboxes showing you what breakpoints have been added to code in the model. You can disable breakpoints by clicking the checkbox next to the line number where they are. You can disable all the breakpoints by unchecking the "Enable Breakpoint Debugging" box. Checking and unchecking these boxes will only affect whether a breakpoint is enabled or disabled, it will not actually delete the breakpoint. To delete a breakpoint, you can highlight it in this window and press the delete key or click on its in the code window. Disabled breakpoints will appear as a in the margin and will not cause the FlexScript execution to stop for debugging.

You may right click on a breakpoint in the breakpoints window to explore the associated code.

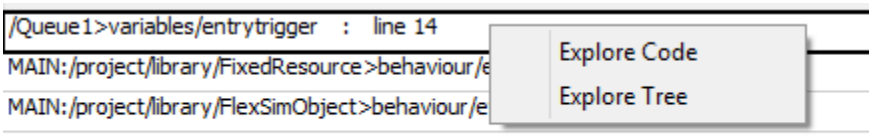

Breakpoints
Enable Breakpoint Debugging
/Queue1>variables/exittrigger
Line 2
/Processor3>variables/cycletime
/Queue1>variables/entrytrigger
Line 8
Line 14
Explore Code

Call Stack

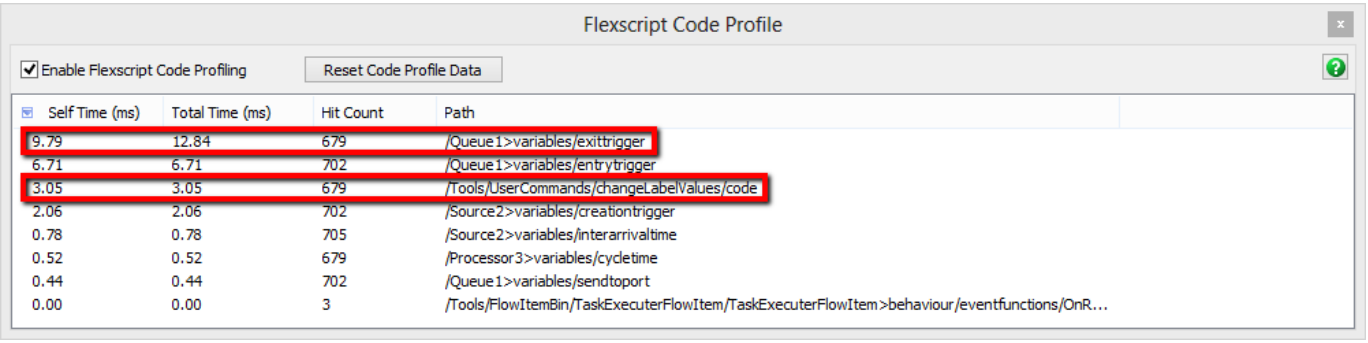


This area shows the current call stack, which is a function call history. The most recent call is always at the top of the list. In this case, the OnEntry function was called by the OnRecieve function, which was called by the OnTimerEvent function.

You may right click on any line of the call stack to explore the associated code or node in the tree.



Code Profiler


Flexscript Code Profile
Enable Flexscript Code Profiling
Reset Code Profile Data
Self Time (ms) Total Time (ms) Hit Count Path
9.79 12.84 679 /Queue1>variables/exittrigger
6.71 6.71 702 /Queue1>variables/entrytrigger
3.05 3.05 679 /Tools/UserCommands/changeLabelValues/code
2.06 2.06 702 /Source2>variables/creationtrigger
0.78 0.78 705 /Source2>variables/interarrivaltime
0.52 0.52 679 /Processor3>variables/cycletime
0.44 0.44 702 /Queue1>variables/sendtoport
0.00 0.00 3 /Tools/FlowItemBin/TaskExecuterFlowItem/TaskExecuterFlowItem>behaviour/eventfunctions/OnR...

The Code Profile window profiles all the flexscript code being executed in your model.

Enable Flexscript Code Profiling- Check this box to enable Code Profiling. Code profiling will remain enabled even if the Code Profile window is closed.

Reset Code Profile Data - Clears the currently accumulated profile data.

Self Time (ms) - This is the total amount of time in milliseconds that the code has taken to execute since the model began.

Total Time (ms) - This is the Self Time plus any time spent calling other functions like User Commands within the code.

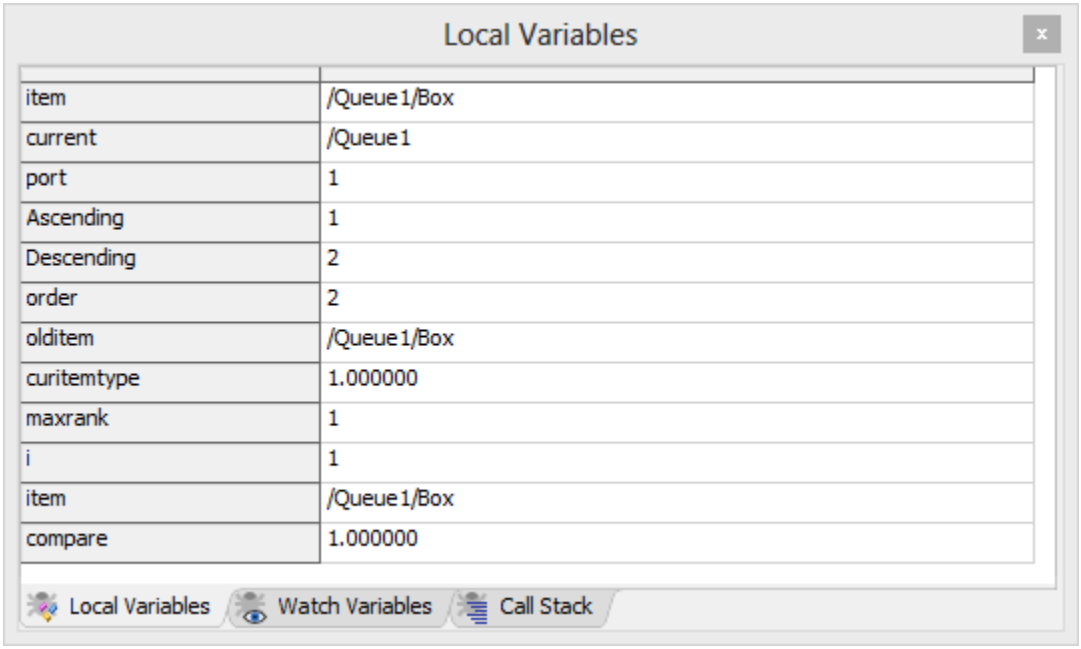
Hit Count - This is the total number of times the code has been executed.

Path - The path to the flexscript node being executed.

Self Time vs Total Time

In most cases the Self Time and Total Time will be equal. However, you'll notice above that the Self Time and Total Time of the Queue1>variables/exittrigger are different. Within the exit trigger, a user command called changeLabelValues is called. The Total Time of the Queue's exit trigger is equal to it's Self Time + changeLabelValue's Self Time.

Local Variables



This area shows you the current values of any locally defined variables. As you step through the code, these values will update immediately so you can see what is happening. Often, models may not behave correctly because variables in code are not what they are expected to be. This window allows you to see exactly what the variables are.

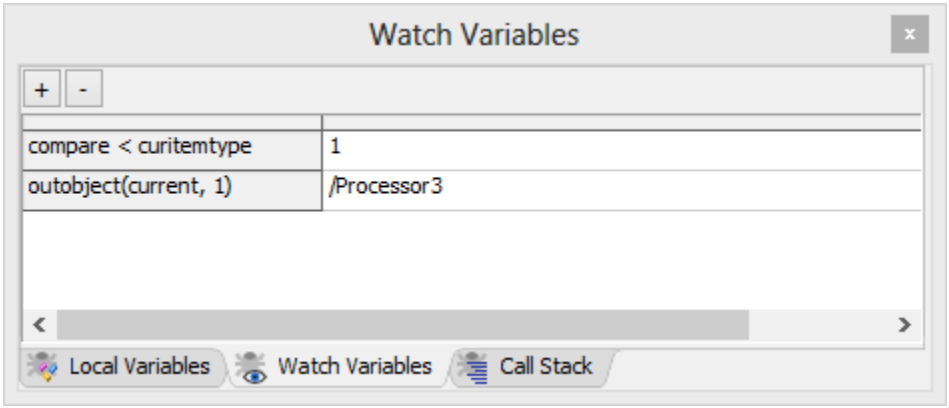
Alternatively, you can mouse over variables in the Code Edit window to see their current value.

```
 8 int Ascending = 1;
 9 int Descending = 2;
10 int order = /** \nSort Order: *//***tag:choice
11 /** \n\nSort entering flowitems in the given o
12 treenode olditem = parnode(1);
13 double c olditem: /Queue1/Box ression: *//***t
14 int maxrank = 1;
15
16 if (curitemtype > 0)
17     debug();
18
19 for (int i = 1; i <= content(current); i++) {
```

Keep in mind, the yellow arrow is pointing to the next line to be executed, so if a new variable is being initialized, you will not be able to see its value until after that line has been executed.

```
19 for (int i = 1; i <= content(current); i++) {
20     treenode item = rank(current, i);
21     double compare = /** \nCompare To: *//***tag:compareto*//**/getitemtype(item)/**/;
22     if (or compare: Out of scope
23         if (compare > curitemtype)
```

Watch Variables

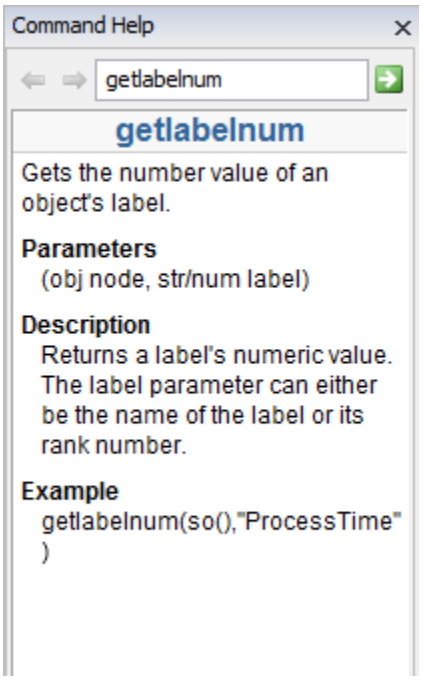


This area allows you to specify other variables or expressions that you want to see, such as global variables. By pushing the + button, you can increase the number of lines in the table. The - button will delete the row that was last clicked in the table. You can double-click on a gray area of the table to enter a variable or expression. Its value will be displayed to the right This can help explain why certain conditional statements, such as used in "if" statements aren't behaving as expected. It also allows you to see global variables that otherwise are not visible on the Local Variables tab.

Command Helper

The Command Helper is a quick reference for flexscript commands. It can be accessed by two methods:

1. From the Help menu.
2. By pressing **F1** while hovering the cursor over a flexscript command.

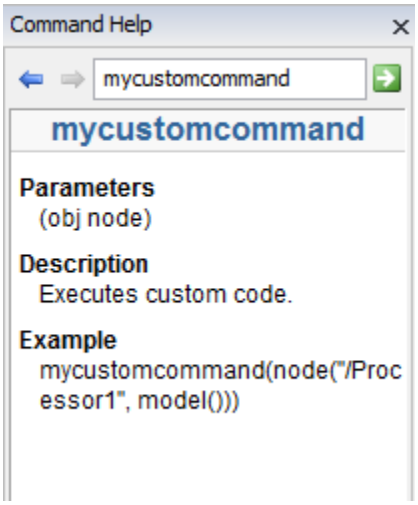

Command Help
getlabelnum
getlabelnum
Gets the number value of an object's label.
Parameters
(obj node, str/num label)
Description
Returns a label's numeric value. The label parameter can either be the name of the label or its rank number.
Example
getlabelnum(so(),"ProcessTime")

- Go to the previous command topic.

- Go to the next command topic.

- Search the command list.

User Commands that the modeler creates will also appear in the Command Helper and in the Command Reference.



Experimenter / Optimizer

1. Experimenter
2. Optimization in FlexSim
3. Example
4. Reference

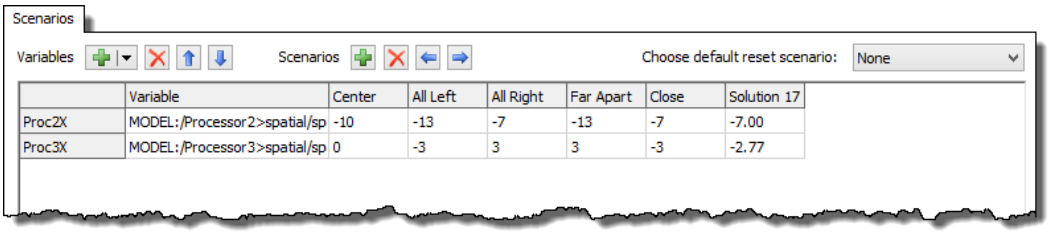
Experimenter

The Experimenter is used to define, run and analyze experiments on defined model scenarios. See the Experimenter Example for an example of how to use the Experimenter.

Topics

- **Variables/Scenarios**
- **Performance Measures**
- **Analyzing Results**

Variables/Scenarios



To create an experiment run, variables are added to the experimenter. These variables are things in the model that you want to change as part of a given experiment. They may be simple values in labels or global tables or they may be the number of Operators for a given team or the position of a FixedResource object. You create and edit Experimenter Variables in the Scenarios Tab. Each variable is a row in the Scenarios table.

Experiment scenarios are associated with experiment variables. A scenario is a specific configuration of the set of variables that you have defined. Variables must have at least one scenario, which is a numeric value that is assigned to the associated variable. In the image above, the scenarios are Center, All Left, All Right, etc. and specify the x position of Processor2 and Processor 3. When the experimenter runs, it will run a defined number of replications for each scenario.

You can have any number of variables and scenarios in an experiment.

Default Reset Scenario

The default reset scenario option allows you to set your model up based up a given scenario. Select a scenario from the drop down and then hit the Reset button to reset the model. The values assigned to the different variable will be applied to the model.

Performance Measures

Performance measures allow you to get statistical data from your model and use it to assess and compare the results of your scenarios. The statistics may include things like throughput, average wait time, or values from dashboard widgets, etc.

Analyzing Results

Once the experiment is finished (all bars in the Experiment Status window are green) you can analyze the results of your performance measures by clicking the **View Results** button of the Experiment Run tab. This will open the Performance Measure Results window.

The data in this window allows you to compare the results of different scenarios. There are several options for how to display the data including a Replications Plot, a Frequency Histogram, a Correlation Plot (for examining correlations between multiple performance measures), a Data Summary, and a Raw Data view.

This window also displays the output of each replication and results from dashboard widgets. Performance measure results and dashboard widgets may then be exported to an HTML format for distribution.

Optimization in FlexSim

What is Optimization?

Optimization begins with a model. For optimization in general, this model can be any system that accepts a set of inputs and then produces outputs. Optimization is the search for a set of inputs that produces the best outputs for a given model.

Within FlexSim, it is possible to optimize a simulation model. The inputs can be thought of as configuration options: How many people should be hired? Where should the storage racks be placed? What kind of machines would work best? The outputs can be thought of as results: What was the utilization of the workers? How long did items spend in the model? What was the total cost per item? FlexSim provides easy-to-use tools that allow you to specify the inputs and outputs for your model. Once this is done, FlexSim can intelligently search through the possible configurations to find the best possible outputs; in other words, it can find an optimal configuration of your model.

Note: This document contains a lot of information. If you are new to optimization in FlexSim, a good learning strategy would be to read this document, then do the optimizer tutorial, then read this document again.

Configurations, Scenarios, and Solutions

In FlexSim, the Simulation Experiment Control tool allows users to define scenarios for a model. A scenario consists of one value for each experiment variable, which results in a configuration of a model. The optimizer also generates configurations for a model based on the same set of variables used by the experimenter. In order to distinguish this kind of configuration from a scenario, it is called a solution. Users generate scenarios, and the optimizer generates solutions. Both configure a model based on the variables defined in the Experiment Simulation Control.

General Algorithm

When the optimizer is running, it is searching for better and better configurations for the model. Here's a basic version of that search algorithm:

1. Generate a model configuration (a solution).
2. Set the current model to match that configuration.
3. Run the simulation. This is also called evaluating the solution.
4. Get the outputs from that run
5. Rank the solution.
6. Generate a new solution based on the results from all evaluated solutions.
7. Repeat from step 2.

The above algorithm repeats until the optimizer runs out of time, evaluates a maximum number of solutions, is stopped by the user, or until it evaluates all possible solutions of the model.

Defining an Optimization

Variables

Optimization variables are the options for your model. When the optimizer generates a solution for your model, it is simply setting each of the variables to a specific value. There are different kinds of variables based on the option that is being represented. There is no limit to the number of variables that an optimization can have.

To add a variable to an optimization, add a variable to an the Scenario Table on the Simulation Experiment Control. If that variable can be manipulated by the optimizer, it will be automatically added to the list of variables for the optimizer. If necessary, the name of the variable will be modified to match the optimizer's variable name requirements.

Type	Description	Examples
Continuous	These variables represent options with a range of possible values. Any value within that range is a valid option for a model (4.354, 6.0, -10.45, etc.).	Values that can be very finely tuned (such as positions, lengths, and times) are often represented as continuous variables.
Integer	These variables represent options with a range of possible values. Any integer value within that range is a valid option for a model (-3, 5, 7, 25, etc.).	Quantities of discrete objects (number of people, items, etc.) are often represented as integer variables.
Discrete	These variables represent options with a range of possible values and a step between possible values. Only values that are exactly *n* steps from the lower bound are valid.	Values with discrete steps (part sizes, paired items, etc.) are often represented as discrete variables.
Binary	These variables represent options that only have two possible values: 0 and 1.	Values that represent options like yes/no, on/off, present/not present, etc., are usually represented as binary variables.
Design	These variables represent options that have a range, but a higher value does not represent "more" or "further." Instead, it just represents a different option. The optimizer will not assume that increasing this kind of variable will have a predictable effect on the system.	Values that represent options like machine type, overall floor layout, or packing strategy are usually represented as design variables.
Permutation	These variables come in groups; a single permutation variable is pointless. Permutation variables can have a value from 1 to *n*, where *n* is the number of permutation variables in a specific group. However, each value is guaranteed to be distinct. If there are three permutation variables in a single group, they can have the values like 1, 2, 3 or 3, 1, 2; they cannot have values like 1, 1, 2 or 3, 1, 3.	These variables are usually used to represent an order. For example, a part route could be represented with a permutation variable. In one configuration, Station 1 could be first, Station 2 second, and Station 3 third. In another configuration, Station 3 could be first, Station 1 could be second, and Station 2 could be third.

Performance Measures

Performance measures are the results of a model run. They usually represent things such as cost, revenue, throughput, risk, average time in system, utilization, etc. There is no limit the the number of performance measures monitored by an optimization.

All performance measures added to the Performance Measures tab in the Simulation Experiment Control are automatically added to the optimizer. If necessary, the name of the performance measure will be modified to match the optimizer's variable name requirements.

Constraints

Constraints are a way of specifying additional conditions that a given model must maintain. Because of the nature of simulation, many constraints are naturally enforced by the model. Occasionally, however, it is necessary to enforce additional relationships.

Constraints are mathematical expressions that result in boolean values. They can be composed of variables, performance measures, literal values, and basic mathematical functions. Examples of valid constraints are shown below:

```
processorPositionX < 30
numberOfEmployees * 500 > 0.5 * totalRevenue
0 < numberOfEmployees + numberOfTrucks < 30
```

If a given configuration breaks one of these constraints, it is marked as infeasible. The optimizer will only mark a configuration as optimal if it is feasible.

A constraint can have a comma-separated list of values; for example:

```
processorPositionX < (30, 35, 40)
```

If a constraint like this exists, the optimizer will basically run the optimization routine for each value in the list, but report the results as one set.

Note: The graphical display for the optimizer may not display correctly if a constraint list exists and multiple objectives are searched. The optimizer will correctly perform the search, but a different utility (such as Excel) will need to be used to visualize the results.

Objectives

Objectives are functions used to determine whether one configuration is better or worse than another. They are very similar to constraints, except they return an actual value rather than a boolean result. Objectives also have a direction; they can be maximized or minimized. For example, an objective function called "cost" might be something like:

```
machineCount * 5000 + throughput * 50
```

If cost were minimized, the optimizer would look for solutions with the lowest cost. If cost were maximized, the optimizer would look for solutions with the highest cost. It is very important to set the correct direction for each objective.

Search Modes

The optimizer has three different search modes. A search mode is the way the optimizer will use the objective function while it runs. The search modes are as follows:

- **Single** - The optimizer will try to maximize or minimize a single objective. The best solution found in the time given will be returned.
- **Weighted** - The optimizer will try to maximize the cumulative value of all given objectives. Each objective is given a weight and a direction. If a particular objective is to be minimized, its value is multiplied by -1 before it is added to the cumulative objective. The best solution found in the time given will be returned.
- **Pattern** - The optimizer will search for solutions that optimize all given objectives. This search mode returns a set of optimal solutions. They are optimal in that for each one, improving any objective worsens the others. For example, a given optimization may maximize profit and minimize cost. For a solution to be optimal, no other solution can exist that increases profit without increasing cost. This is also called "Pareto Optimal."

There are more options for weighted searches, including an objective min, max, and goal. These are not well documented, even by OptQuest; use at your own discretion. Leaving blank or at zero in FlexSim will make sure they are ignored.

Other Optimization Options

Because the optimizer must run a simulation to evaluate each solution, there are many options about those simulations that can be set, such as:

- The model run time per solution evaluation
- The model warmup time per solution evaluation
- The number of replications per solution
- The kinds of data that should be saved per replication

There are also many options about the optimizer itself that can be set, such as:

- The wall time - the amount of real time the optimizer will spend searching for an optimal solution
- The maximum number of solutions to evaluate
- The use of experimenter scenarios as solutions.

Running an Optimization

Once the optimization has been properly defined, the hard part is done. You can click the **Optimize** button on the Optimizer Run tab of the Experiment Simulation Control. At this point, FlexSim will use the same multi-threaded capability found in the experimenter to evaluate multiple solutions simultaneously. This allows FlexSim to rapidly search through generated solutions, as well as provide meaningful feedback about the progress of the optimizer.

Interpreting the Results

When the optimizer is finished, one (or several) solutions will be marked as optimal solutions. Unless the optimizer evaluated all possible solutions (which is generally impossible), then the optimal solutions are simply the best solutions that the optimizer found. They are **not** guaranteed to be the absolute best solutions. They are only guaranteed to be better than all other solutions that were evaluated.

The true value of the optimizer, however, will be missed if you only look for those few optimal solutions. As the optimizer searches for optimal solutions, it creates a database of each solution and its effects on the model outputs. This database can be used (with the proper application of statistical techniques) to find previously unknown relationships between model variables, or to determine which constraints were most restrictive. Using this data appropriately can lead to new insight into the behavior of the model.

Under the Hood

FlexSim uses the OptQuest engine from OptTek Systems, Inc. OptQuest is a trusted, industry-standard optimization engine that uses the latest evolutionary algorithms to effectively search complicated design spaces. More information on optimization with this engine can be found in the OptQuest documentation here.

Experimenter/ Optimizer Example

Introduction

This tutorial introduces the experimenter and optimizer found in FlexSim. These tools each help to answer the "what-if" questions for a model: What if the milling station were placed differently? What if more people were hired for a given shift? What if the order of these processes was changed? The experimenter allows you to try many different versions of your model and evaluate the performance of each one. Alternatively, the optimizer can automatically search through many different versions of your model, looking for the best one. They are both very powerful tools that enable you to learn about and improve your system.

Note: Using the optimizer requires the OptQuest license. Contact FlexSim for more information about obtaining this license.

What You Will Learn

This tutorial will walk you through the following steps:

- Creating an experiment
- Running an experiment
- Viewing experiment results
- Designing an optimization
- Running an optimization
- Viewing optimization results

Along the way, we will cover the definitions of related terms, such as:

- Experiment variables
- Performance measures
- Scenarios
- Optimizer variable types
- Constraints
- Objectives
- Solutions

Step-By-Step Model Construction

For this tutorial, let us examine a very simple situation. A single worker must carry an item from a source to a processor. After the item finishes processing, the worker must carry it to a second processor that takes longer than the first. After the the second processor finishes the item, the worker then carries it to the sink.

Now let us suppose we want to maximize the throughput (which is also tied to revenue) of this system by adjusting the position of the processors. If each processor could be moved up to three meters right or left, where should each be placed? It would be very difficult to intuitively know how to place both processors to maximize throughput. In order to solve this problem accurately, we will use the experimenter and optimizer. Now, obviously this is a drastically simplified scenario, but often in real life you have situations where you want to see how various layouts affect overall throughput. This is a very simplistic implementation of such an experiment.

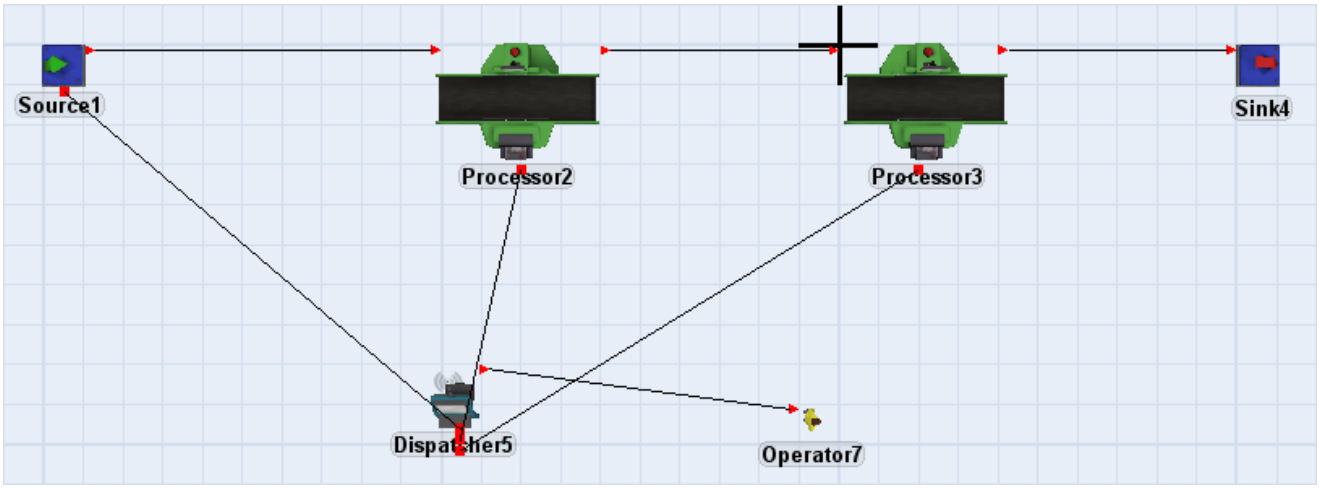
Step 1: Building the Model

Model

Create a new model using **Seconds**, **Meters**, and **Liters** for units.

Objects

Create a **Source**, two **Processors**, a **Sink**, a **Dispatcher**, and an **Operator**. Lay these objects out as shown below.



Positions

Set the location of the objects according to the table below:

Object	X Position	Y Position
Source1	-20.0	0.0
Processor2	-10.0	0.0
Processor3	0.0	0.0
Sink4	10.0	0.0

Dispatcher5	N/A	N/A
Operator7	N/A	N/A

Dispatcher5 and **Operator7** do not need to be in a particular place, but they should not be in line with the rest of the objects.

Logic

Set the following logic:

- Set **Source1**, **Processor2**, and **Processor3** to **Use Transport** (available in the Quick Properties menu).
- Set the process time of **Processor2** to `normal(10, 2)` (also available in the Quick Properties menu).
- Set the process time of **Processor3** to `normal(12, 3)`.
- Set the reset position of **Operator7** to its current position.

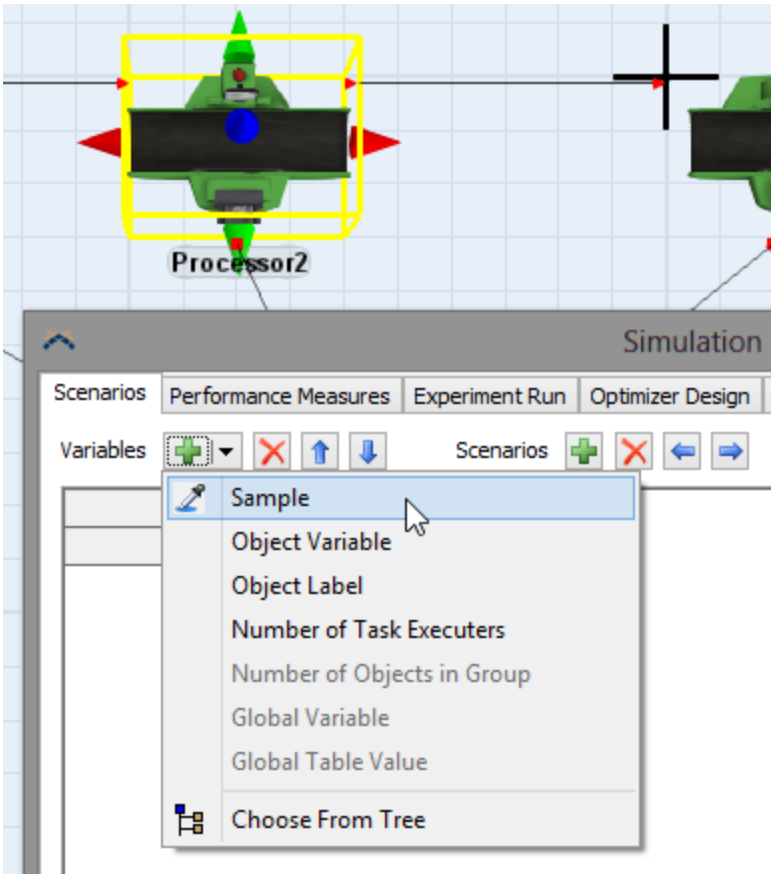
Step 2: Defining the Experiment

The remainder of this tutorial deals with using the Experimenter, found in the Statistics menu. The optimizer uses most of the functionality already present in the experimenter.

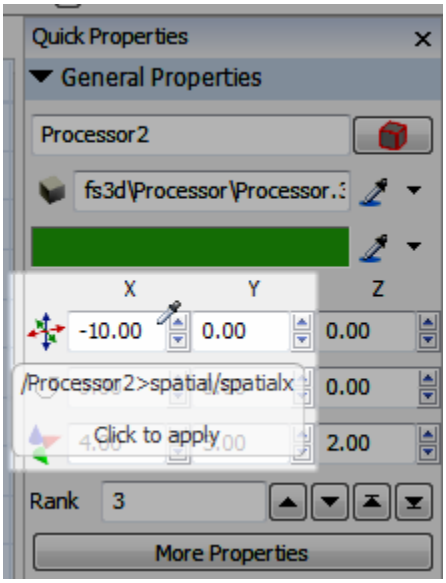
Creating Variables

Open the Experimenter window. Position the window so you can see the processors in the model as well as the window. Then, for Processor2 and then Processor3, follow these steps:

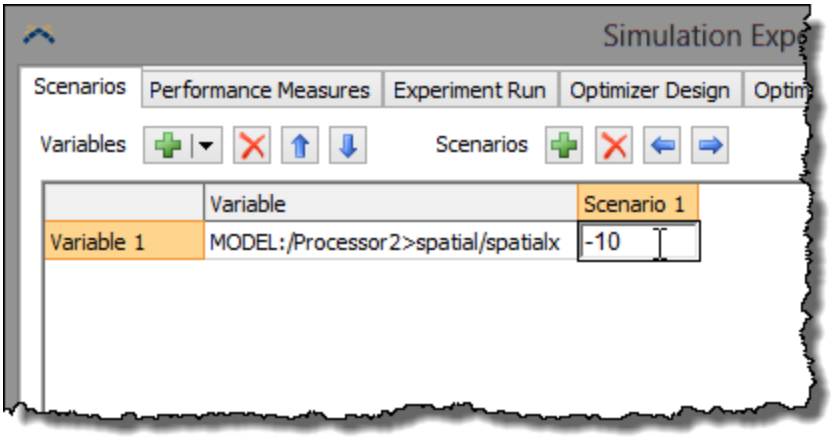
- Click on the processor in the 3D view.
- Click the down arrow on the button.
- Select **Sample** from the popup menu. This puts the cursor in sample mode.



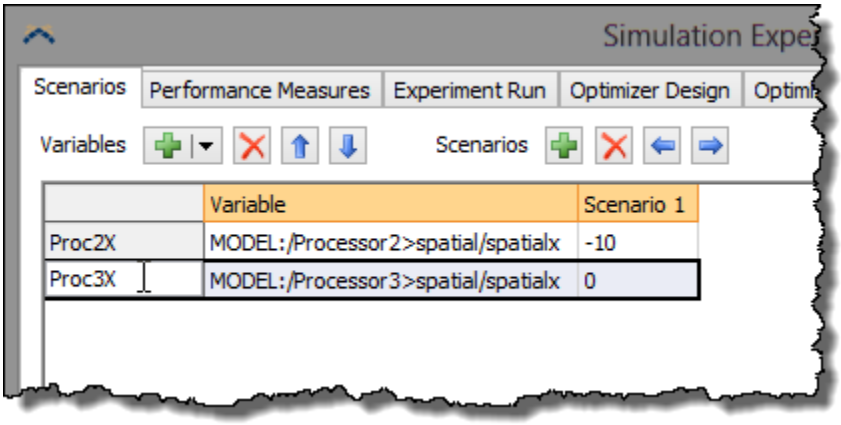
- Sample the X position field in the Quick Properties menu by clicking on it. This adds a new experiment variable.



- Set the value of Scenario 1 for the new variable by double-clicking the cell and entering the new value.



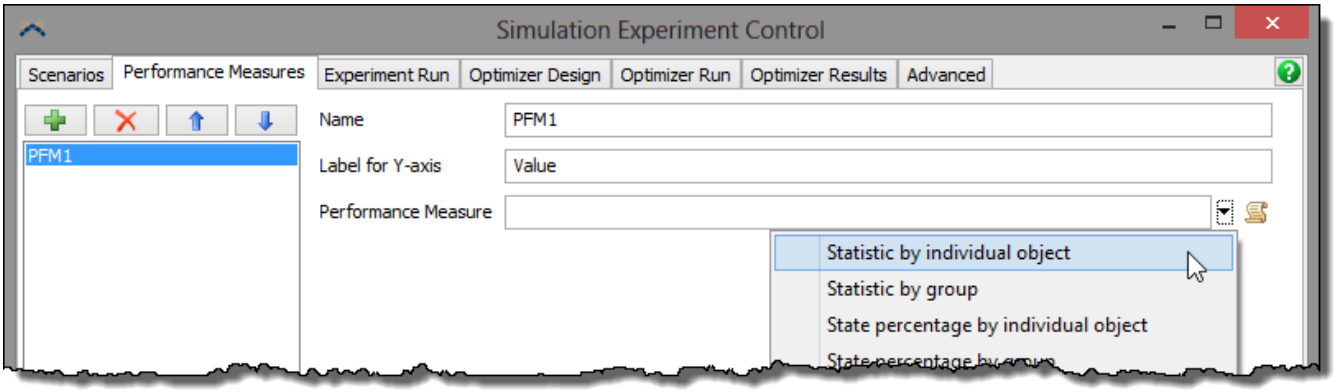
- Set the name of the variable by double-clicking on the current name. Set the name to **Proc2X** for **Processor2** and **Proc3X** for **Processor3**.



Creating Performance Measures

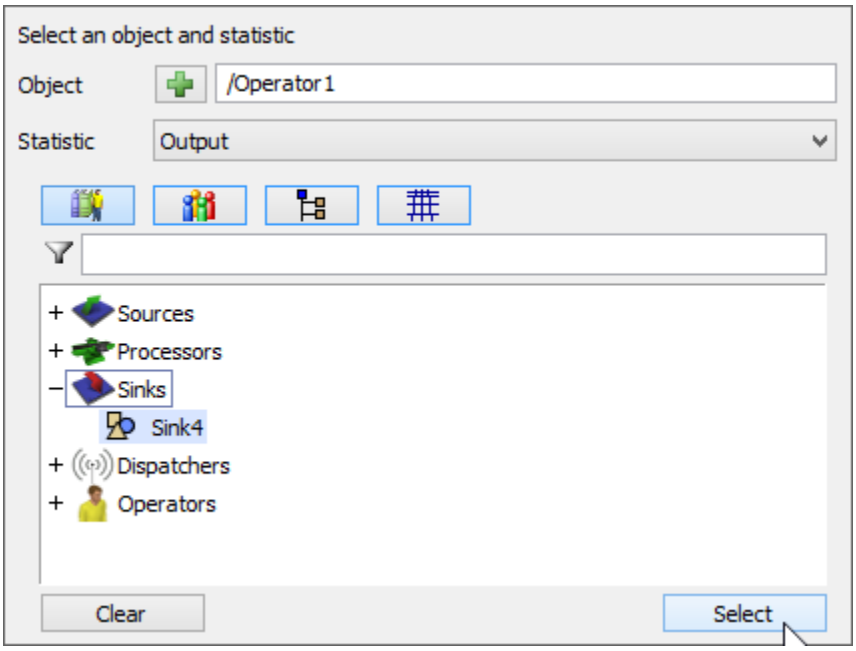
Go to the Performance Measures tab in the Experimenter window. From there:

- Click the button to add a new performance measure.
- Name the new performance measure **Throughput.**
- Click the ▾ button and select the first option. A popup will appear.



- Select **Sink4** for the object and **Input** for the statistic. To select the object simply type "/Sink4" (no quotes) in the object field or do the following:
  - Click the button.
  - Select **Sink4** from the list of objects.

- Click the **Select** button when you are finished.



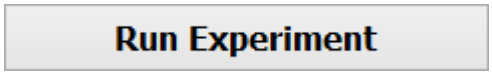
Designing the Experiment

Now that we've created the variables and performance measures, we'll set up some scenarios for our experiment. Go back to the experiment tab:

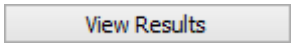
- Create 5 scenarios by clicking on the Scenarios button 4 times.
- Enter scenario names and values as follows:

	Variable	Center	All Left	All Right	Far Apart	Close
Proc2X	MODEL:/Processor2>spatial/spatialx	-10	-13	-7	-13	-7
Proc3X	MODEL:/Processor3>spatial/spatialx	0	-3	3	3	-3

Running the Experiment

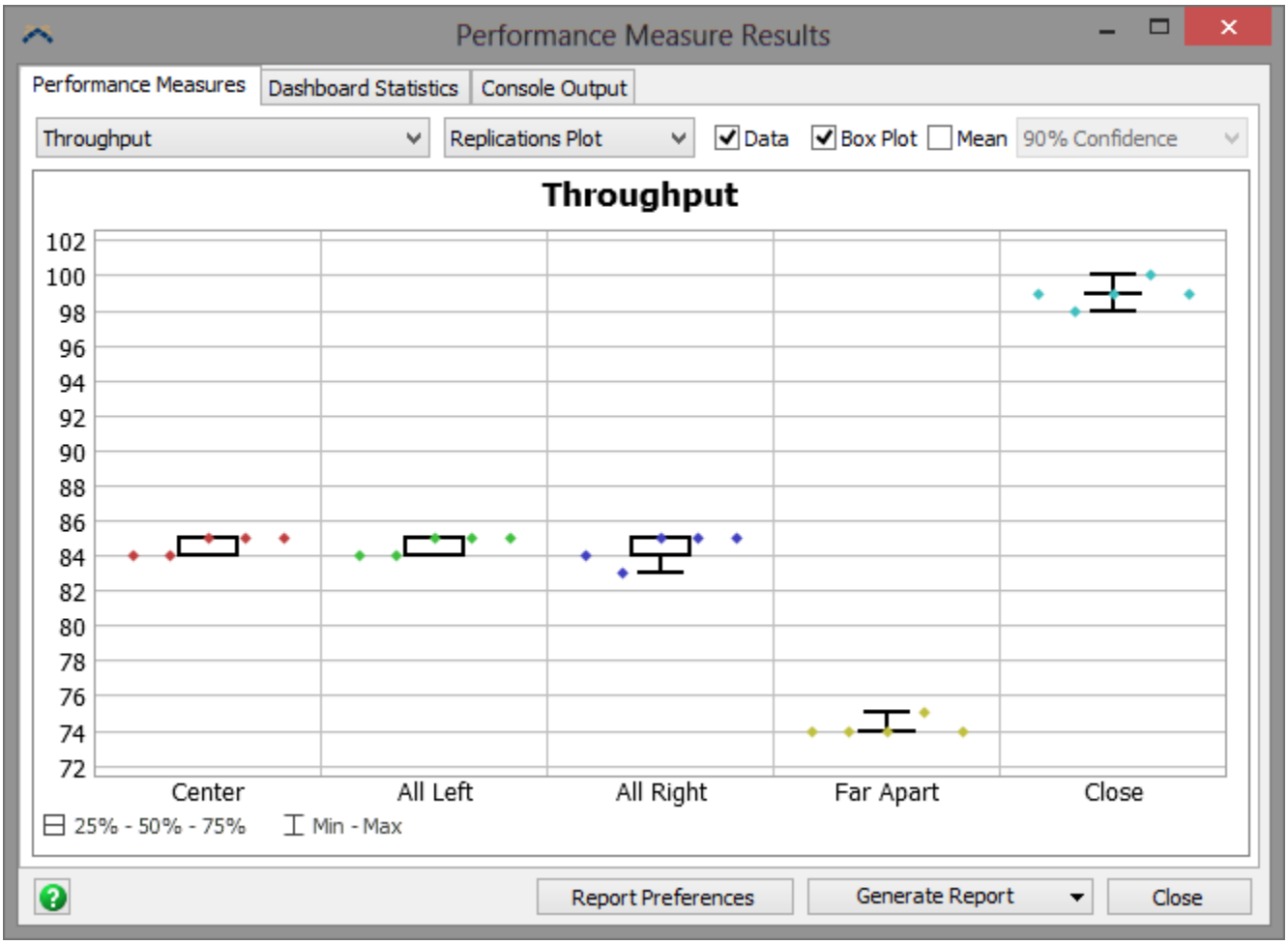
Go to the Experiment Run tab. Hit the 

button. Each scenario will be run 5 times and the results of the Throughput performance measure will be collected at the end of each run. The status window will show which scenarios/replications are currently being run. FlexSim will run multiple scenarios simultaneously if your computer has a multi-core cpu.

Once the experiment is finished, click the 

button at the bottom. This will open a window where you can get data on the performance measures for the scenario. In this example we only have one performance measure, but if you had multiple you could see the results for each in this window. There are several options for how to display the data, including a Replications Plot, a Frequency

Histogram, a Correlation Plot (for examining correlations between multiple performance measures), a Data Summary, and a Raw Data view.



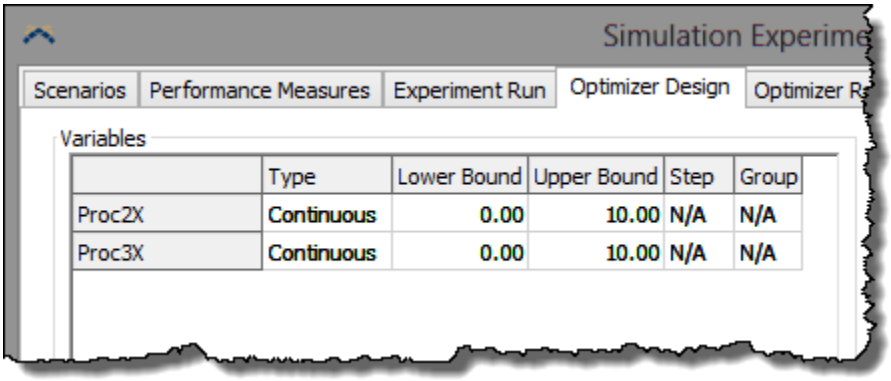
In this experiment, the best scenario was the "Close" scenario, which averaged right around 99 parts of throughput. The worst scenario was the "Far Apart" scenario, which averaged about 75 parts throughput.

Optimization

In addition to using the Experimenter to explicitly define scenarios, you can use the Optimizer. The optimizer will automatically create scenarios and then test those scenarios, trying to find a scenario that best meets an objective.

Designing the Optimization

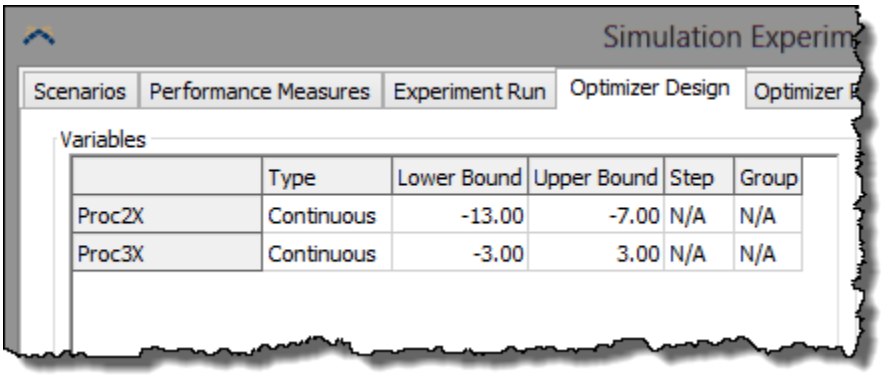
Go to the Optimizer Design tab in the Experimenter window. You will see that the two variables created earlier are present; this is because the experimenter and optimizer share variables.



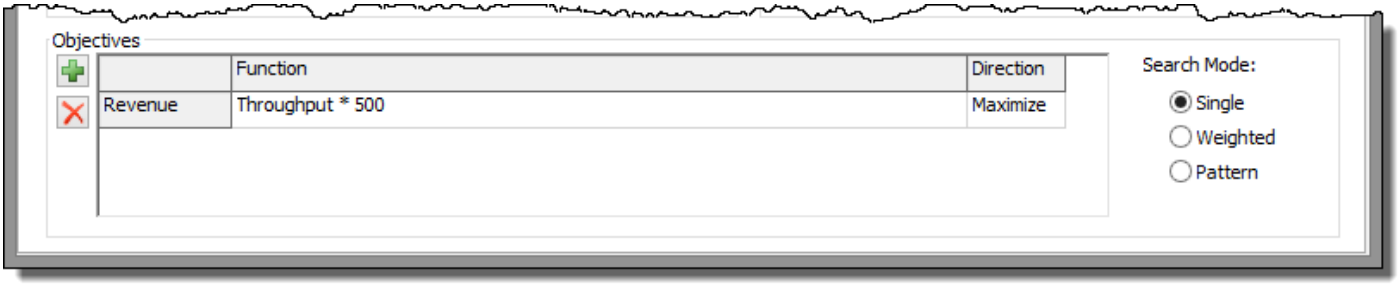
However, the optimizer needs additional information about those variables. Specifically, you must specify:

- **Type** - The type of a variable dictates what kinds of values are possible for a given variable. Continuous variables can have any value between the upper and lower bound.
- **Lower Bound** - The lower bound specifies the lowest possible value the optimizer can set for the variable.
- **Upper Bound** - The upper bound specifies the highest possible value the optimizer can set for the variable.
- **Step** - For Discrete and Design variables, the step specifies the distance between possible values, starting from the lower bound.
- **Group** - For permutation variables, the group specfies which set of permutation variables this particular variable belongs to.

For this optimization, we want to allow the processors to move three meters to either side. Since we are not limited to specific positions within that range, both position variables are **Continuous**. However, we need to set the lower and upper bounds of each variable. To edit values in the table, double-click on the cell of interest and enter in the new value. Enter in the values shown below:



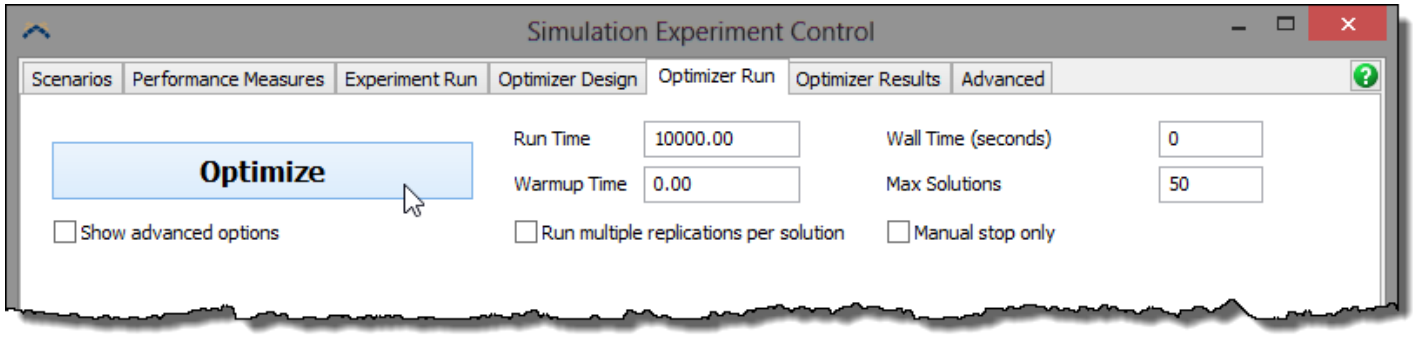
The final design step is to set the objective function. The objective function is present, but blank. Edit its name to be **Revenue**. Then click on the function field. A button will appear on the right side. Click on this button to bring up a list of all variables and performance measures. The objective function is a value derived from any or all of these values. Select **Throughput**; this will add that perfomance measure to the objective function, and put the cursor right and the end. Add the text `* 500` so that **Revenue** is equal to `Throughput * 500`. Leave the direction on **Maximize**, because we want to maximize **Revenue**. Since we only have one objective, the search mode can remain on **Single**.



Step 3: Running the Optimization

Go to the Optimizer Run tab in the Experimenter window. Then:

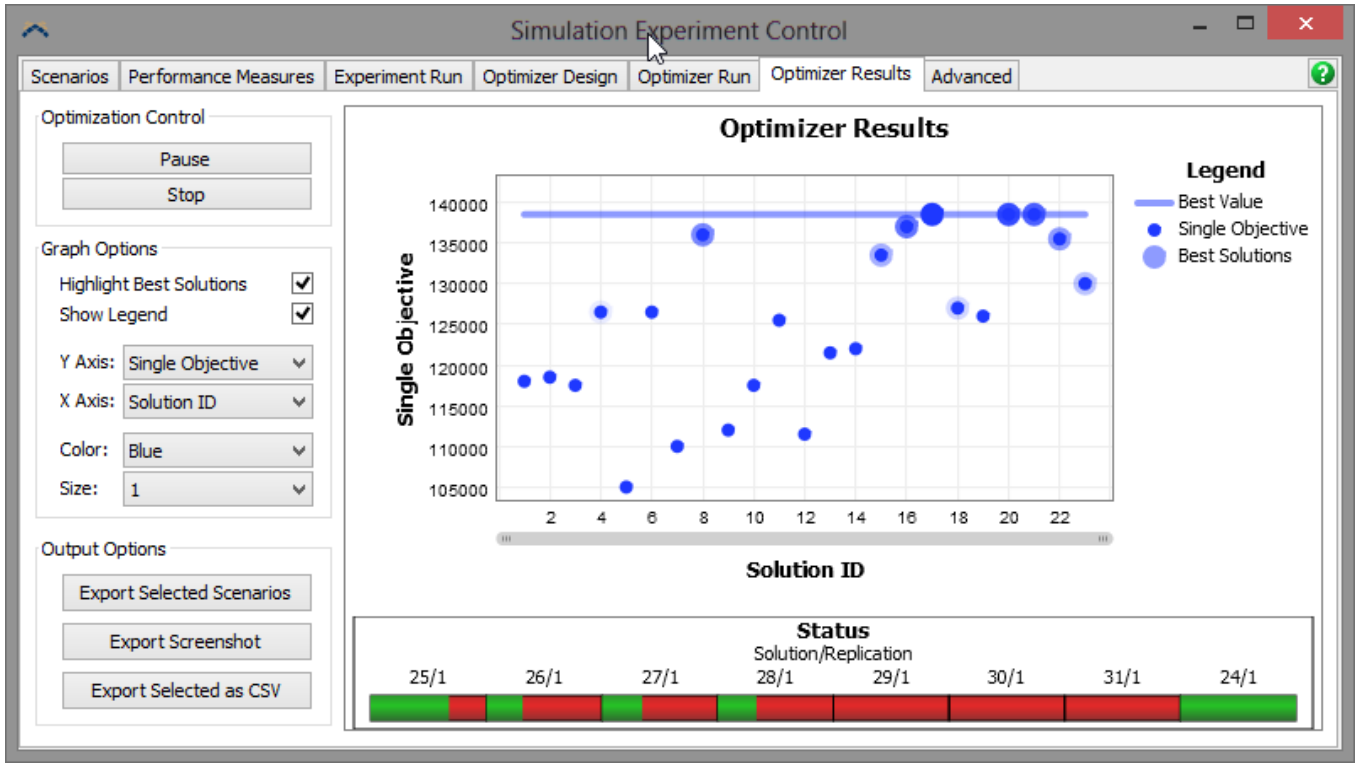
1. Set the **Run Time** to 10000. This is how long the optimizer will run each model configuration (called a solution) to evaluate it.
2. Set the **Wall Time** to 0. This usually means how long the optimizer is allowed to run in real time. The value 0 means it has no time limit.
3. Set **Max Solutions** to 50. This means the optimizer will try no more than 50 different solutions to find the optimal solution.
4. Click the **Optimize** button.



The Experimenter window will automatically switch to the Optimizer Results tab. The optimizer then begins running through the following loop:

1. Determine values for **Proc2X** and **Proc3X**.
2. Run a model with those values for 10000 seconds.
3. Evaluate the performance measures.
4. Calculate the objective function.
5. Rank this solution.
6. Use the information from this solution to create a new solution - new values for **Proc2X** and **Proc3X**.
7. Repeat from Step 2.

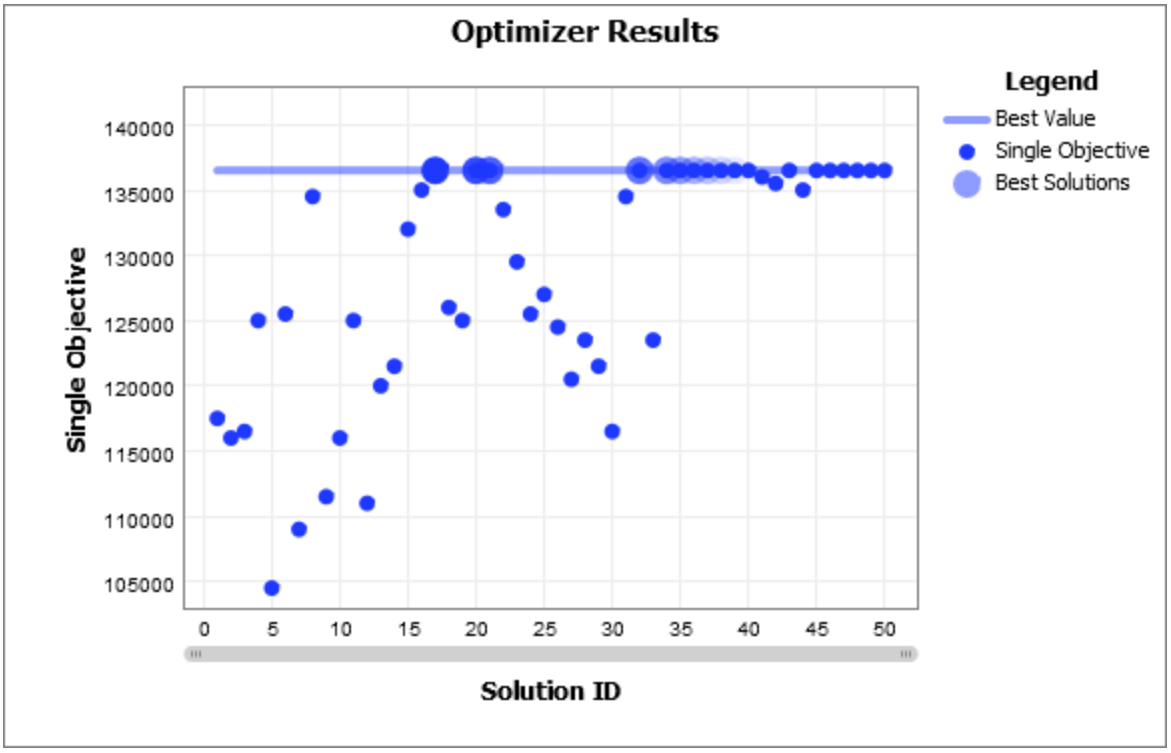
Because the optimizer shares the multi-threaded capability of the experimenter, it can evaluate multiple solutions at the same time. As the optimization progresses, the Optimizer Results graph will update and show the optimizer's progress.



Once the optimizer evaluates 50 solutions, a message will appear stating why the optimizer stopped. In this case, it will say that the optimizer reached the maximum number of solutions. If something went wrong, the message will contain information about the error.

Step 4: Analyzing the Results

When the optimization is finished, the Optimizer Results chart should look something like this:

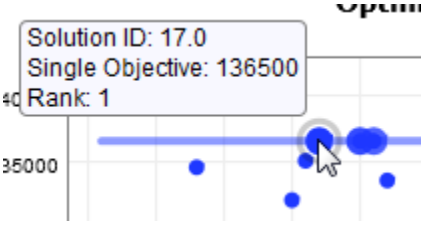


The Y Axis is called "Single Objective." For this example, it is synonymous with **Revenue**. The best solutions are highlighted. The circles with a lighter border around them represent better solutions. For a single objective, the top 10 solutions are marked this way.

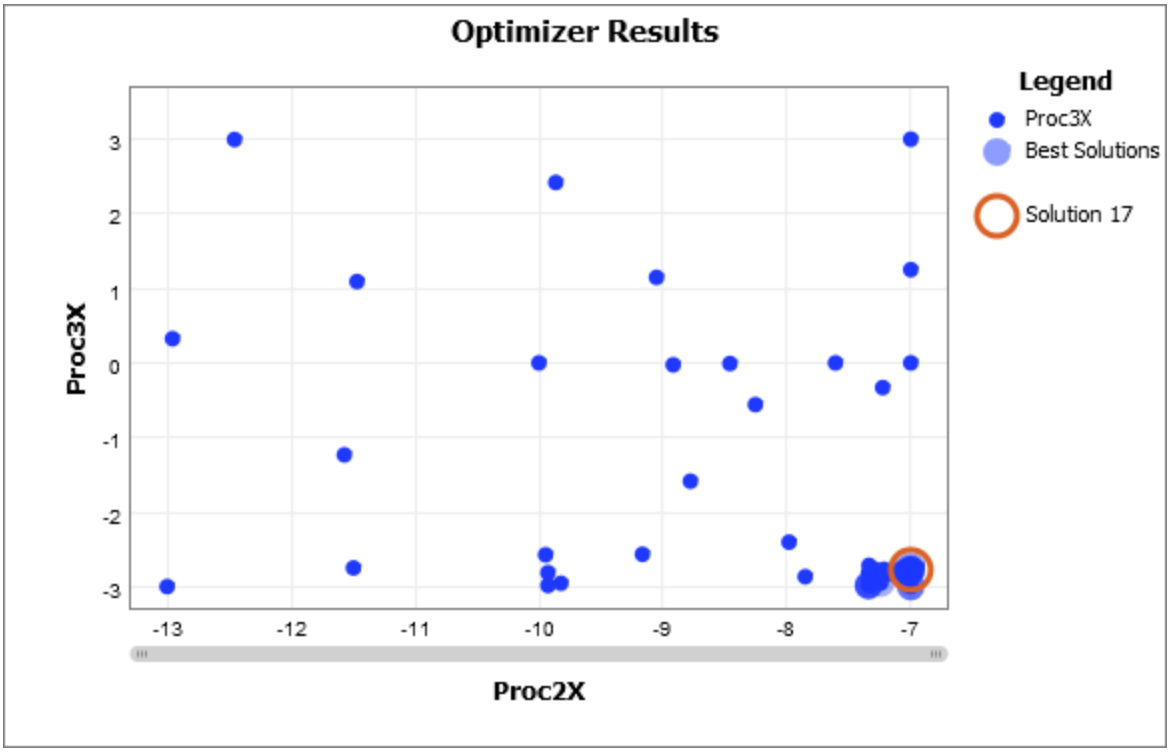
As the optimization progressed, the optimizer got better and better at creating good solutions, so that the last 15 solutions were all very good. This is called convergence, and it is one way to tell if an optimization is finished; if the objective value has not improved for a while, it may be that it will not improve with further searching, and the current best solution should be used.

Answering the Original Question

The goal of this optimization was to figure out where to put the two processors. We can now very easily find the answer to this question. Hover over the best solution (the largest blue circle) on the chart; a small popup will appear.



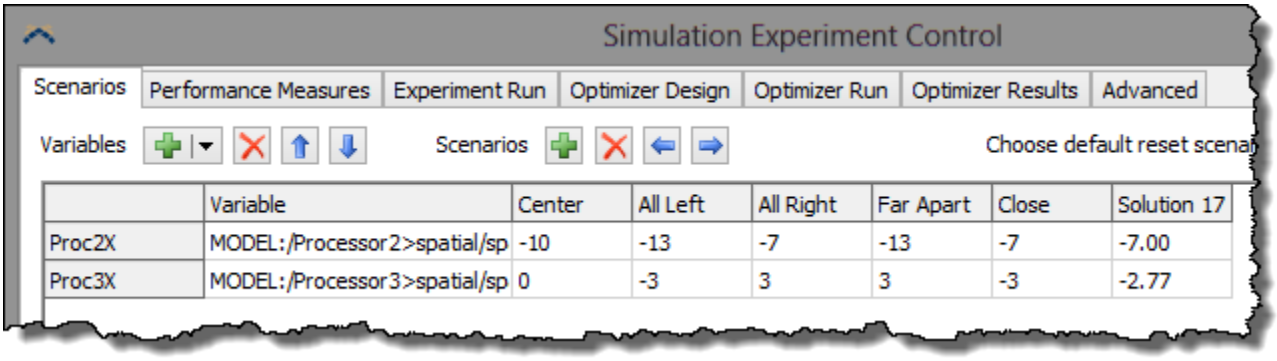
Click on this solution to select it. Now, in the Graph Options panel, change the Y Axis to Proc3X, and the X Axis to Proc2X.



The best solution (and all the other best solutions) is found where Proc2X is greatest, and where Proc3X is least. Remember that all top 10 solutions produced the same results; in this case, having the two processors right next to each other is the best configuration for this model.

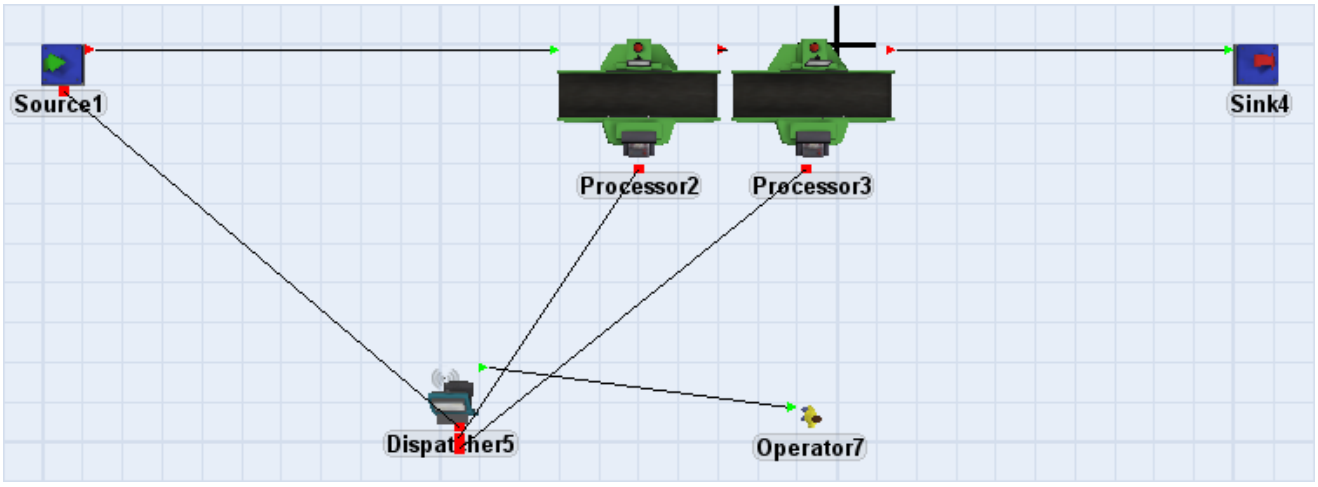
Setting the Model to the Best Solution

It can be very useful to set the model to match the best solution. To do this, click the **Export Scenarios** button. This takes all the selected solutions and creates experimenter scenarios for them. Go back to the Scenarios tab on the Experimenter window to view the new solution.



	Variable	Center	All Left	All Right	Far Apart	Close	Solution 17
Proc2X	MODEL:/Processor2>spatial/sp	-10	-13	-7	-13	-7	-7.00
Proc3X	MODEL:/Processor3>spatial/sp	0	-3	3	3	-3	-2.77

Now, from the "Choose default reset sceanrio" drop-down on the far right, select the new scenario. Then reset the model to apply those values to the 3D model.


Source1
Processor2
Processor3
Sink4
Dispatcher5
Operator7

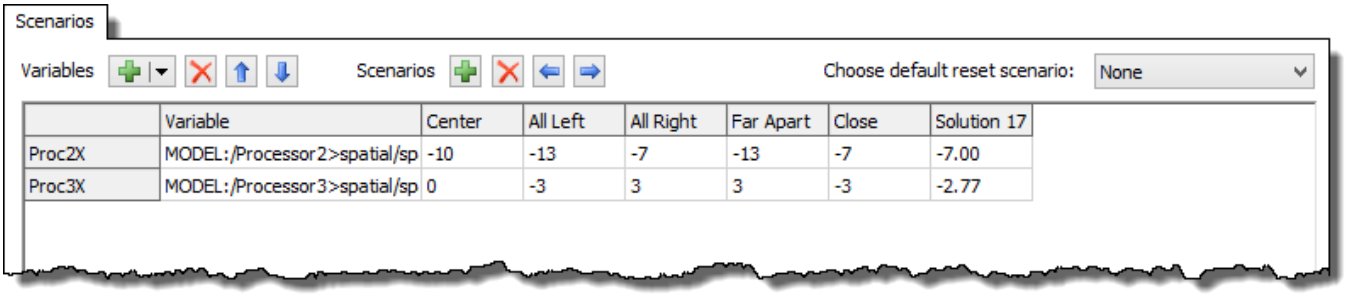
Experimenter / Optimizer Reference

Pages

- **Scenarios**
- **Performance Measures**
- **Experiment Run**
- **Performance Measure Results**
- **Optimizer Design**
- **Optimizer Run**
- **Optimizer Results**
- **Advanced**

Scenarios

The Scenarios tab is where you define the variables and scenarios associated with an experiment.



Variables

- Add an experiment variable. Select the variable type from the drop down.

- Remove the selected variables.

- Move the selected variables up or down in the list.

Once variables have been added, you can change them by clicking in the "Variable" cell for that variable. The buttons will appear. Use these to change the variable, or type in a path manually.

Define the name of the variable in the header column of the table on the left side.

Scenarios

- Add a scenario. Once added, enter the value for each variable in that scenario's column in the table.

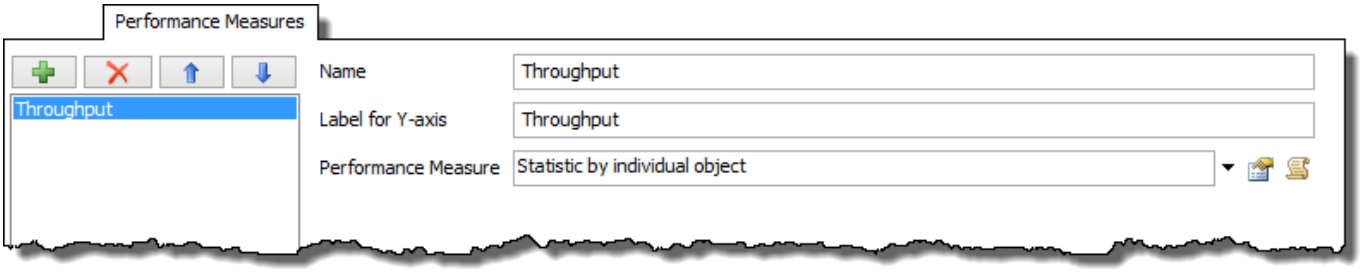
- Remove the selected scenarios.

- Move the selected scenarios left or right in the list.

You can rename the scenarios by changing the name of the table's column header.

Choose Default Reset Scenario - This allows you to define a scenario for your model to go to when you reset your model outside the experimenter. Once chosen, whenever you reset, the model will go back to that scenario. Choose None if you don't want the model to set its scenario when reset.

Performance Measures



- Add a Performance Measure. If there are dashboards in the model, you can select a statistic from a dashboard as your performance measure.

- Remove the selected performance measure.

- Move the selected performance measure up or down in the list.

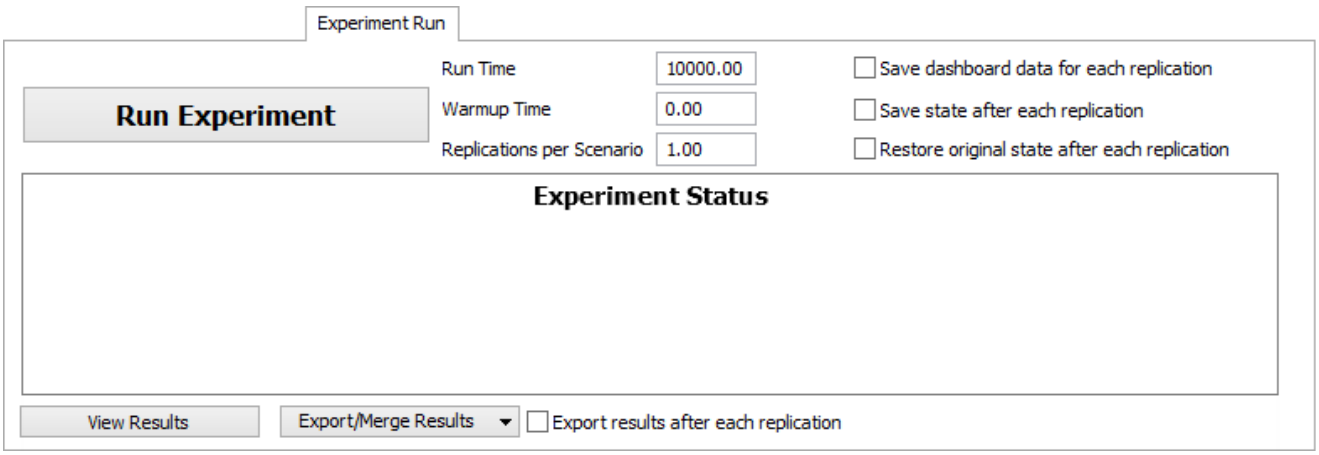
Select a performance measure from the list to edit it. If it is a standard performance measure, controls will display on the right (as seen in the above image). If the performance measure is associated with a dashboard statistic, a **Properties** button will appear. This will open the dashboard widget's properties window.

Name - The name of the performance measure will be used in the Performance Measure Results window and can be used in the Optimizer Design tab when defining Constraints and Objectives.

Label for Y Axis - Defines the Y axis title in the Performance Measure Results window.

Performance Measure - This picklist specifies what information will be gathered. Choose from the available picklist options or write your own custom code.

Experiment Run



The Experiment Run tab is where you define parameters for the experimenter and run it.

Run Experiment - Starts the experiment run.

Run Time - The total simulation time that each experiment will run to.

Warmup Time - The simulation time that each replication will run to before resetting their statistics. Statistics will thus only be collected for the time period (Run Time - Warmup Time).

Replications Per Scenario - The number of replications that will be run for each scenario.

Save Dashboard Data for Each Replication - If checked then at the end of each replication FlexSim will save the data for each dashboard statistic in the model so they can later be viewed as part of the results.

Save State After Each Replication - If checked, each replication's full simulation state will be saved to a file at the end of the replication. This allows you to open the replication in the state where it finished. This can be especially useful if one of your replications fails to give valid results.

Restore Original State After Each Replication - If checked, FlexSim will completely reload the model between execution of each model. You might check this box if your model doesn't properly reset to the same exact state every time you reset, and you don't want that "spill-over" state affecting subsequent replication results. However, because it is completely reloading the tree, it may increase the time it takes to run each replication.

Export results after each replication - Saves results to a file after each replication.

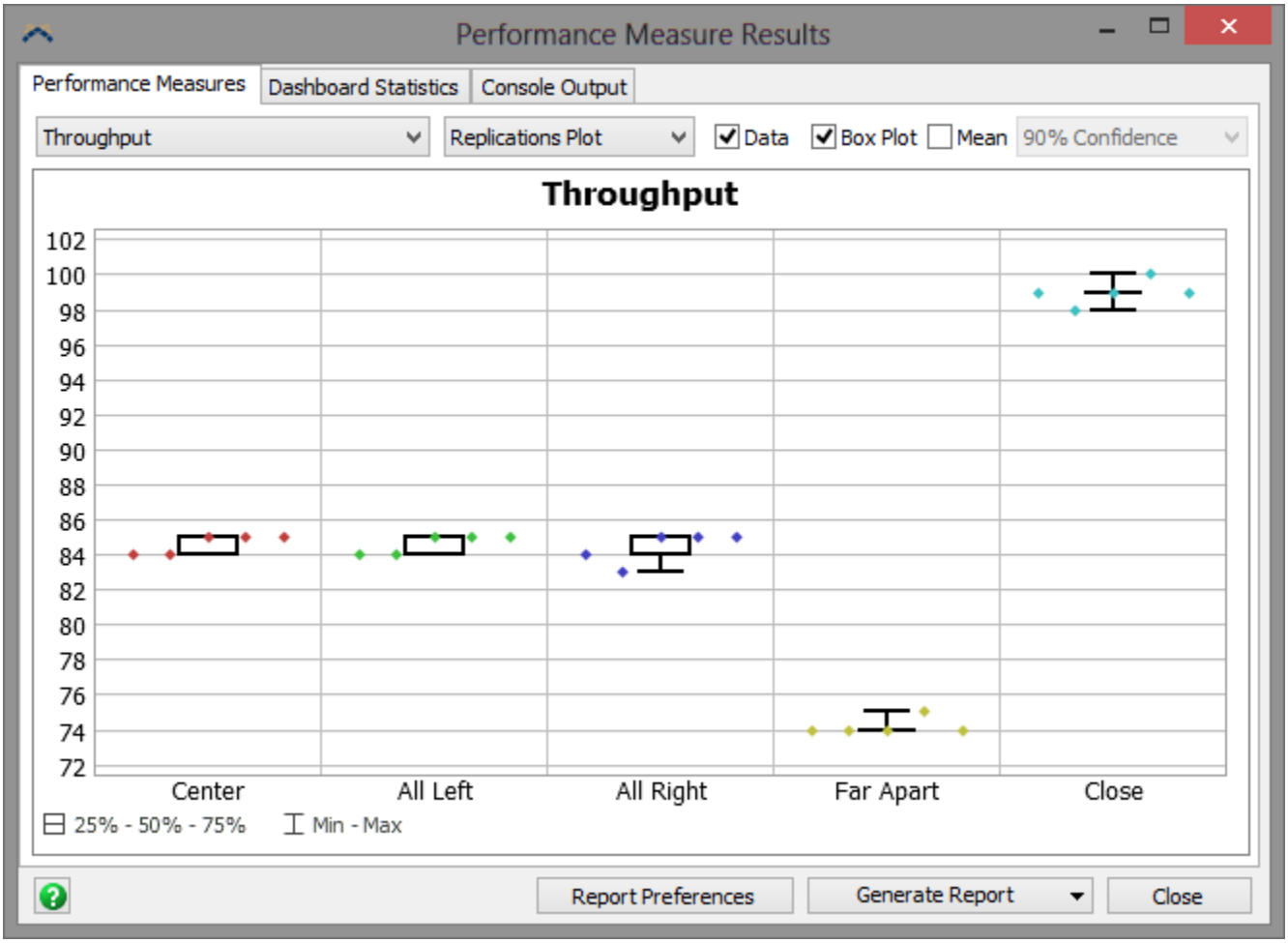
View Results - Opens the Performance Measure Results window. Click this button once an experiment run has finished.

Export / Merge Results - This button allows you to save your result to a file, as well as load / merge results from a saved file into your current results.

- **Export** will save the results to a .t file.
- **Load** will load the results from a saved file into your model, replacing any current results.
- **Merge** will load results from a file, and then merge those results with your model's current results.

Export results after each replication - Saves results to a file after each replication.

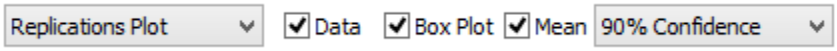
Performance Measure Results



Performance Measures Tab

This tab allows you to compare performance measures between the different scenarios. Select the Performance Measure and graph type. The graph types available are: Replications Plot, Frequency Histogram, Correlation Plot, Data Summary, and Raw Data.

Replications Plot

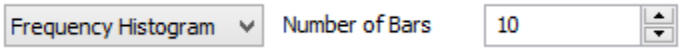


Data - If checked, the replications plot will plot the data points for all replications.

Box Plot - If checked, the replications plot will display a box plot.

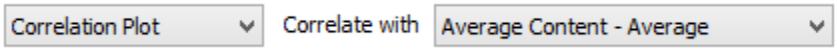
Mean - Check the box to display the mean confidence interval for each scenario. You can specify the percentage as 90%, 95% or 99% Confidence.

Frequency Histogram



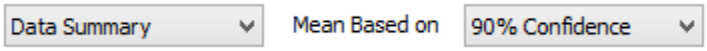
Number of Bars - Specifies the size of range to display.

Correlation Plot



Correlate with - Select another performance measure to plot the current performance measure against.

Data Summary



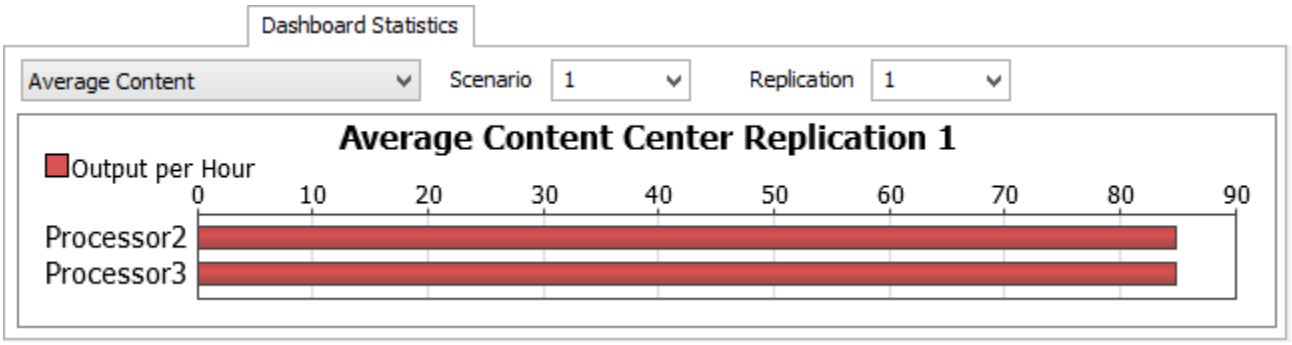
Mean Based on - Displays values based on the selected confidence interval.

Report Preferences - Opens the Performance Measures Report Preferences window. This allows you to prune what data you want included in the report.

Generate Report - There are two options for generating reports:

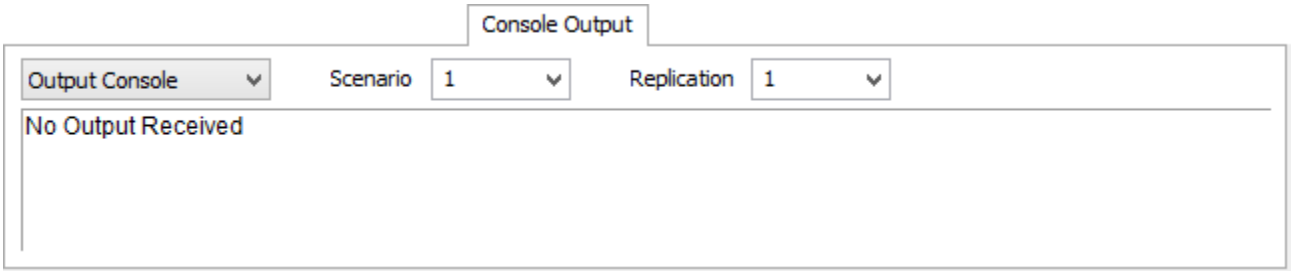
- **Report Format** This creates an html file with full results for all performance measures.
- **Web Viewer Format** This creates an html file that is interactive. When you open the html file you can choose which graphs you'd like to view.

Dashboard Statistics Tab



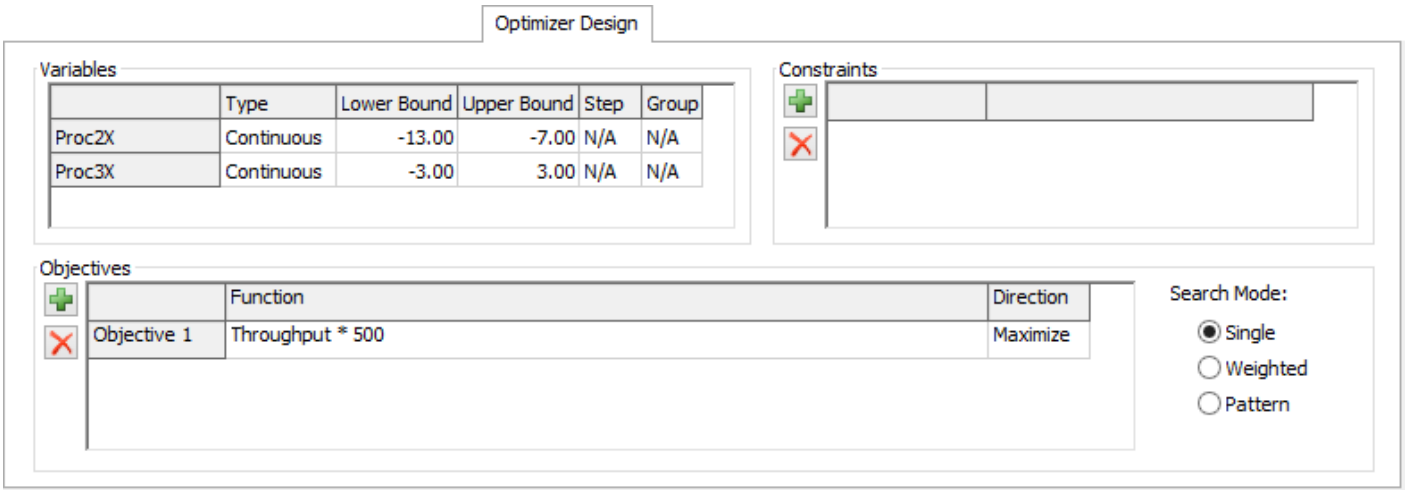
If you have have checked the box *Save Dashboard Data for Each Replication*in the Experiment Run tab before running the experiment, then you can go to the Dashboard Statistics tab to view individual replication results for various dashboard statistics. Choose the statistic, replication and scenario, and the statistic's associated graph will be shown.

Console Output Tab



FlexSim also saves the text output of the Output Console and the System Console for each replication. This can be used for debugging/analysis of each replication. Choose Output or System Console and the scenario and replication you want to view.

Optimizer Design



Variables

Variables Table - This table displays the variables that are listed in the Scenarios tab. Click in one of the Type cells to display a ▾ to change a variable's type. Setting the type will change the cells of the table to reflect which options are applicable.

Constraints

- Add a constraint.

- Remove the selected constraint.

Constraint Table - This table displays all of the constraints for the optimizer. Define an equation using the available variables, boolean and mathematical operators.

Objectives

- Add an objective.

- Remove the selected objective.

Objectives Table - This table displays all of the objectives for the optimizer. Define a function using the available variables, boolean and mathematical operators.

Search Mode

There are three search modes:

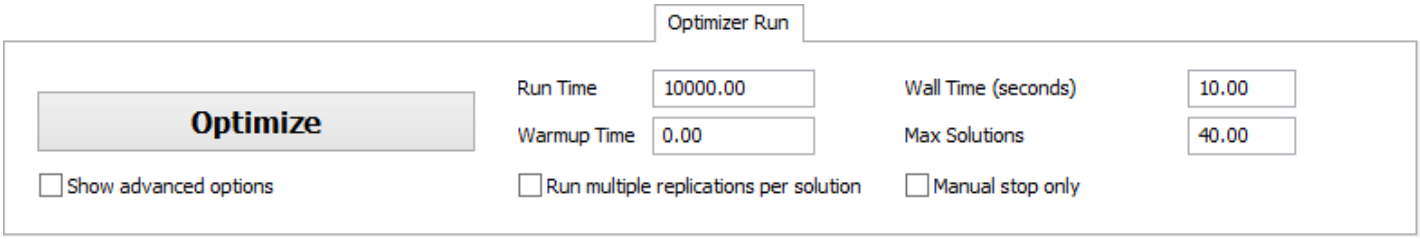
- *Single* - The optimizer will try to maximize or minimize a single objective. The best solution found in the time given will be returned.
- *Weighted* - The optimizer will try to maximize the cumulative value of all given objectives. Each objective is given a weight and a direction. If a particular objective is to be minimized, its value is multiplied by -1 before it is added to the cumulative objective. The best solution found in the time given will be returned. Selecting weighted will change the objectives table to display the following:

	Function	Weight	Goal	Direction	Lower Bound	Upper Bound
Objective 1		0.00	0.00	Maximize	0.00	0.00

- *Pattern* - The optimizer will search for solutions that optimize all given objectives. This search mode returns a set of optimal solutions. They are optimal in that for each one, improving any objective

worsens the others. For example, a given optimization may maximize profit and minimize cost. For a solution to be optimal, no other solution can exist that increases profit without increasing cost. This is also called "Pareto Optimal."

Optimizer Run



Optimize - Begins the optimization process.

Run Time - This is the time in model units that each solution will run.

Warmup Time - The simulation time that each solution will run to before resetting their statistics. Statistics will thus only be collected for the time period (Run Time - Warmup Time).

Wall Time (seconds) - This is the maximum real time the optimizer will spend generating solutions. Once the wall is hit, the active solutions will finish running and then the optimization run will stop.

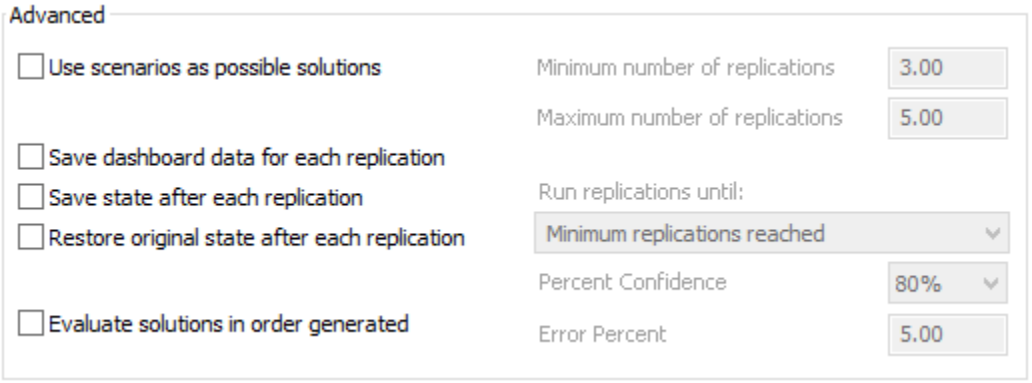
Max Solutions - The maximum number of solutions the optimizer will generate and test.

Run multiple replications per solution - If checked, the optimizer will run multiple replications for each solution that is generated.

Manual stop only - Sets the optimizer to run until the user stops it.

Advanced Options

Check **Show advanced options** to display the following:



Use scenarios as possible solutions - Use manually-defined scenarios (from the Scenarios page) as initial search points. This gives the optimizer a place to begin it's optimization process and generally shortens the optimization time.

Save dashboard data for each replication - Check to save data from dashboard statistics at the end of each replication for post-optimization reports/analysis.

Save state after each replication - Check to save the full state of the model to a file at the end of each replication.

Restore original state after each replication - Check to restore a clean copy of the model after each replication.

Evaluate solutions in order generated - Check to ensure solutions are evaluated in the order they are generated. This is slower, but ensures that the optimization search is repeatable.

If *Run multiple replications per solution* is checked, the following options are made available:

Minimum number of replications - The minimum number of replications that will always be performed for each solution.

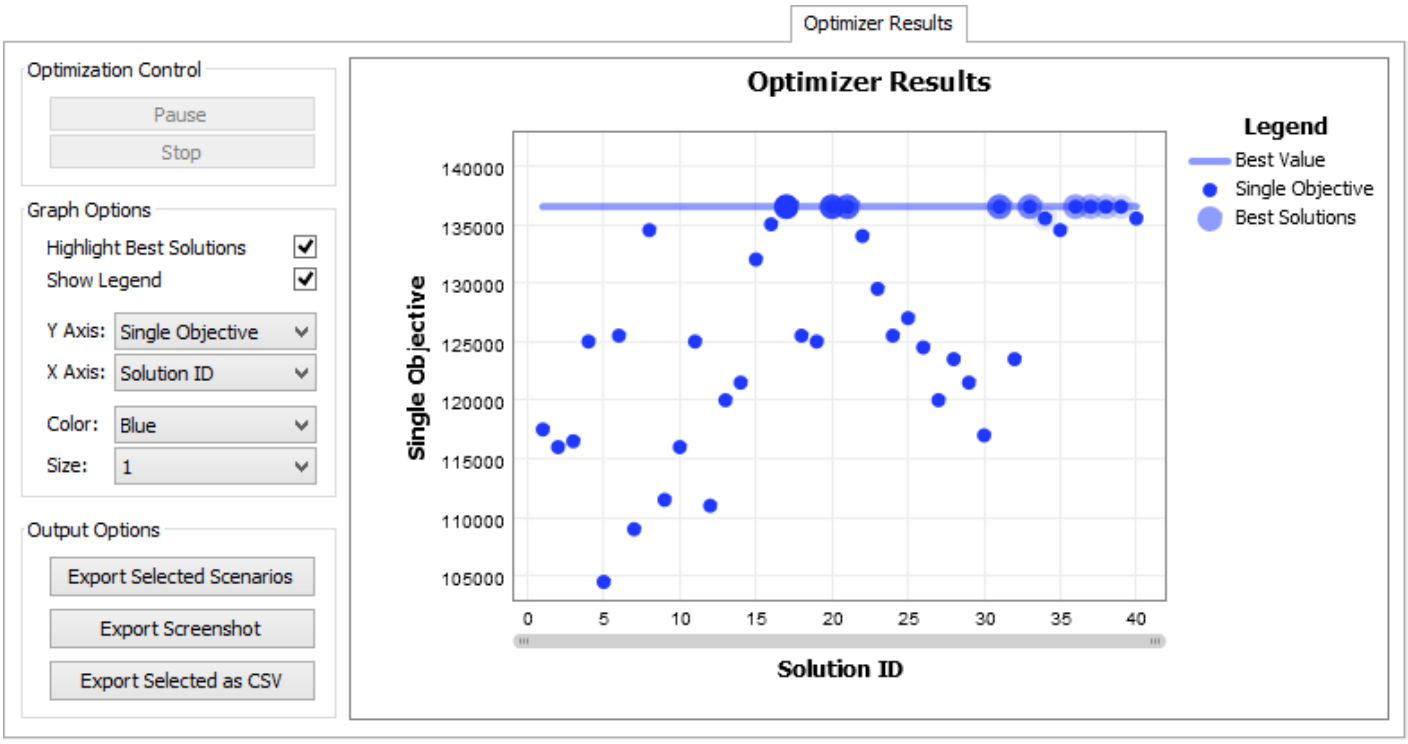
Maximum number of replications - The maximum number of replications that will always be performed for each solution. If set to -1, no maximum will be used.

Run replications until - Each solution will run replications until the chosen condition is met or the maximum number of replications is reached (if not set to -1).

Percent confidence - Percent confident that the solution falls within the specified error percent of the mean value of the replications.

Error percent - Determines the range about the mean for which the percent confidence applies. Must be a value between 0 and 100.

Optimizer Results



Optimization Control

If the optimizer is currently running through scenarios, you can pause or stop the optimization process.

Graph Options

These options allow you to customize the look of the graph and define what values to plot against the X and Y axes.

Output Options

Export Selected Scenarios - This option will take the selected scenarios (click on a scenario in the graph to select) and export their data to the scenarios table of the Scenarios tab. This allows you to either run further experiments on the chosen scenarios, or to set your model to one of the selected scenarios by using the *Choose default reset scenario*.

Export Screenshot - Saves the currently displayed results graph as a PNG file.

Export Selected as CSV - Exports the data from the selected scenarios to a CSV file.

Optimizer Results Graph

Optimization Control - If the optimizer is currently running through scenarios, you can pause or stop the optimization process.

Advanced

For more information see the Experimenter Picklists.

Repeat Streams of Replication - Specify a replication to set the manual model runs (not experimenter replications) to use the same random number streams. You must be repeating random streams for this to take effect.

3D Media

1. Importing 3D Media
2. Preparing a 3D File
3. Importing AutoCAD Drawings
4. Shape Factors
5. Shape Frames
6. Level Of Detail

Importing 3D Media

FlexSim can import several types of 3D media. These include:

- .wrl, .3ds, .dxf, .stl, .skp, .dae, .obj, .ac, .x, .ase, .ply, .ms3d, .cob, .md5mesh, .irr, .irrmesh, .ter, .lxo, .csm, .scn, .q3o, .q3s, .raw, .off, .mdl, .hmp, .scn, .xgl, .zgl, .lvo, .lvs, .blend

Note on importing wrl files: FlexSim will only import VRML version 1.0 shapes, not version 2.0.

Note on importing stl files: FlexSim will only import stl ascii files, not stl binary files.

You can import media using three methods:

1. The most common method is to import media by selecting a 3D file from an object's General Properties of the Quick Properties window.
2. Through the object's General tab page.
3. If there are some 3D media files that are not used by any model objects by default, but you want to change their 3D representation dynamically during the simulation run to use these 3D media files, you can use the Media Files window from the Tools Menu to import media files explicitly.

The following will help to ensure media is imported correctly:

- **Shape Factors**
- **Letting Color Show Through the 3D Shape**
- **Level of Detail**

Preparing a 3D File

We suggest using AC3D for 3D media creation. A license is relatively cheap and AC3D has a lot of capability coupled with a decent UI that's pretty quick to learn. Another large advantage is, FlexSim is capable of directly importing the raw .ac files created from AC3D. You can get AC3D from www.inivis.com.

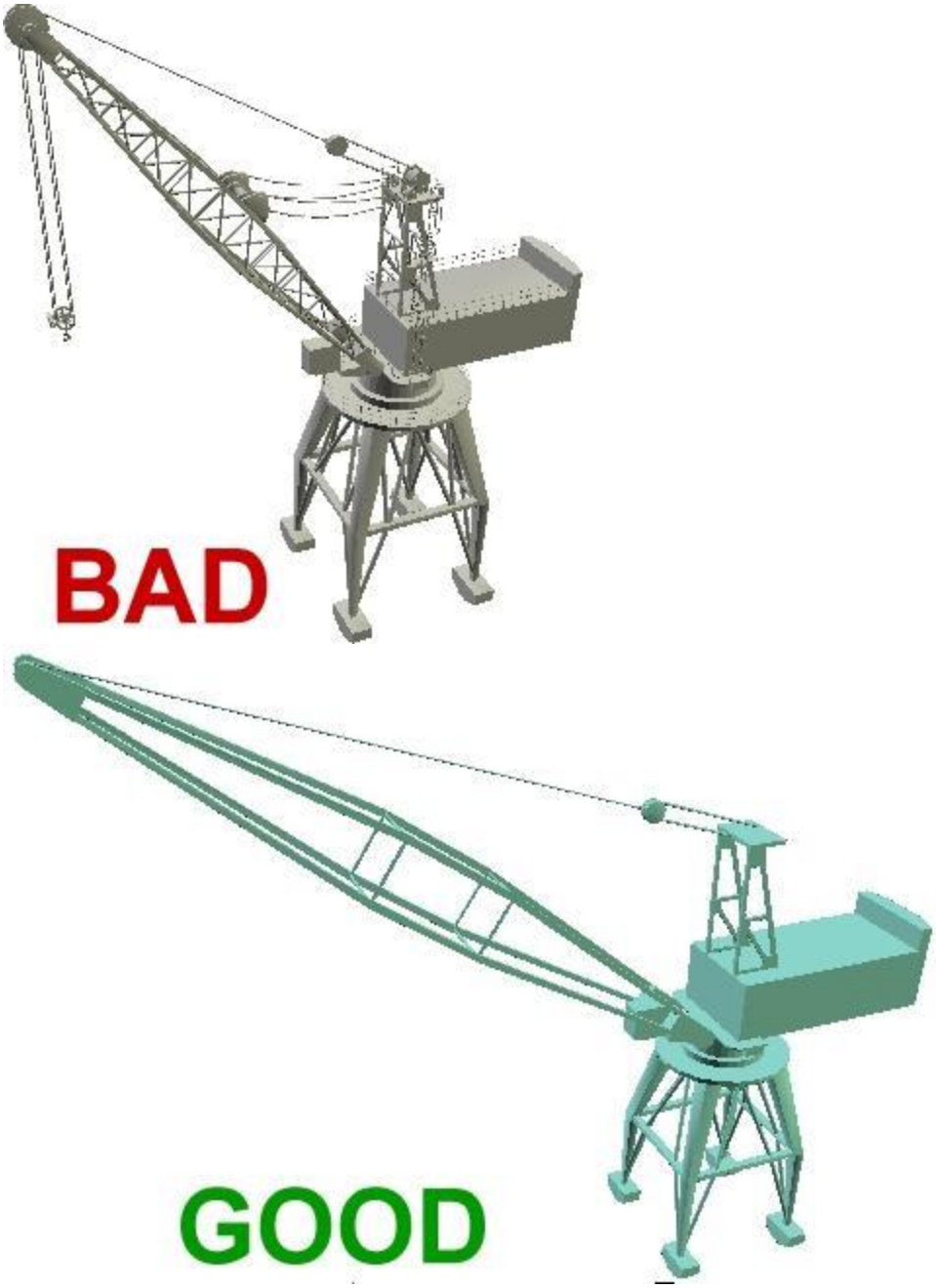
If you're not using AC3D, we suggest exporting media to the 3ds file format. The 3ds file format allows for very fast drawing speeds and fast load speeds. With several 3D formats, including .3ds and .ac files, you can:

- Have the FlexSim object's color show through on certain parts of the shape
- Use the FlexSim object's defined texture show through on certain parts of the shape
- Auto-scale the FlexSim object based on 3D file dimensions

The following steps should be performed when preparing a 3D file for import:

Reduce the Number of Polygons

3D files typically include more information than is necessary. Removing excess polygons will improve the visual performance of your model. As a result, it will be easier to build and present the model.



Textures vs. Polygons

The use of well made textures can help a modeler reduce the number of polygons required to make a realistic looking object. Below are some pictures of low polygon 3D files in FlexSim. Also, where possible you should consolidate objects in your 3D creation software and share the same texture among multiple objects. This reduces the number of OpenGL state changes required, which can significantly improve performance.

Transparency

Transparency in FlexSim is simple. It is just a matter of using textures that have transparency in them. You'll need to create your textures with a file type that supports transparency like a PNG file. FlexSim will read the transparency of the image and display it properly.

Adjust the Scale

3D files are not necessarily drawn in feet or meters and may need to be rescaled to work appropriately in FlexSim. There are three ways to adjust the scale of a 3D file:

1. Appropriately scale the file in the 3D program.
2. Scale the visual tool or other object in FlexSim that the file is imported into.
3. Use an xrl file for wrl files.

If you are using 3DS or .ac, then you can just make sure the object is scaled properly in whatever 3D software you use. Then import it into FlexSim, and it will automatically import to the correct size. Just make sure that your units in your software are the same as the units you are going to use in FlexSim.

Getting the Object's Color to Bleed Through

For .3ds and .ac files, you can have the FlexSim object's color bleed through on certain shapes in the file. You can do this in one of two ways: (1) give the shape an ambient color value of rgb: (0.235,0.235,0.243) in your 3D software, (2) append "_fsclr" to the end of the shape's material name in your 3D software.

Getting the Object's Texture to Bleed Through

For .3ds and .ac files, you can also have the FlexSim object's defined texture show up on the shape instead of the texture defined in the file. To do this, just add a texture named "fstx.png" to the object in your 3D software.

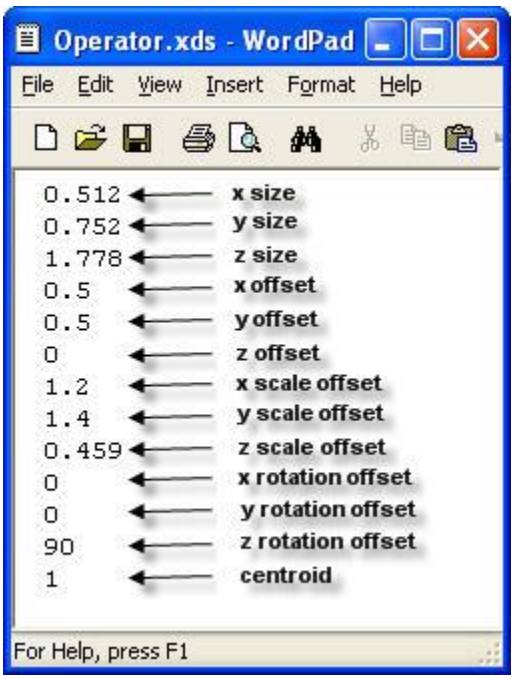
XRL Files

xrl files are used to make imported wrl shapes conform to the object they are imported into. xrl files must have the same names as the objects they modify. For example:

- crane.wrl – crane.xrl

xrl file makeup

The xrl file is a text file made up of 13 values separated by carriage returns. You can edit this file using notepad or wordpad, as shown below.



xrl file info

Spatial values determine the end size of the 3D shape. Offset values are the values required to get the 3D shape zeroed out and sized to 1,1,1. The centroid value is 1 or 0 and determines if the object rotates around the center of the object or the top left corner.

Importing AutoCAD Drawings

The following steps should be performed when preparing an AutoCAD .dxf or .dwg file for import:

1. **Remove all unnecessary information:** AutoCAD files typically include much information that is unnecessary to the simulation. Typically, all a simulation needs is a basic layout. Removing information that is extraneous to the simulation will make your model more clear and reduce the burden on your graphics card. As a result it will be easier to build and present the model. Remove any parts of the drawing that are not pertinent to the simulation study.
2. **Adjust the scale to FlexSim units:** AutoCAD files are often scaled in inches. FlexSim models are often scaled in feet or meters. It is important that the AutoCAD file be rescaled to work appropriately in FlexSim. For instance, to convert from an AutoCAD file in inches to a FlexSim model in feet, the scale factor will be 1/12. To determine how much to scale, follow these steps:
    1. Measure a known distance in AutoCAD ("_dist").
    2. Apply the following equation: scale factor = FlexSim distance / ACAD distance.

    To scale objects in AutoCAD follow these steps:

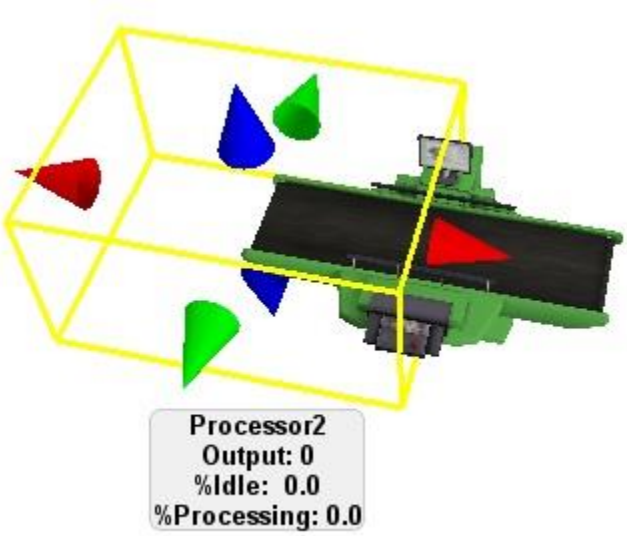
    3. Select the objects you want to scale.
    4. Type "_scale" in the command prompt or select the scale command from the menus.
    5. Specify a reference point.
    6. Type in the calculated scale factor in the command prompt.
3. **Move objects to the origin:** AutoCAD drawings are usually drawn using a specific coordinate system. This usually means that the objects are not located near the origin (0,0,0). When a .dxf file is imported into FlexSim, it is positioned in FlexSim's coordinate system according to the .dxf's positioning. So if the origin point of your AutoCAD file is very far away from the actual drawing, when that .dxf file is imported into FlexSim, the layout will also be very far away from the model's origin position in FlexSim. This can be quite frustrating for the modeler. For this reason, move your AutoCAD objects to the origin. Do so by completing the following steps:
    1. Select the objects you want to move.
    2. Type "_move" in the command prompt or select the move command from the menus.
    3. Specify a reference point.
    4. Type in the desired location of that point in the command prompt.
4. **Explode compound objects.** FlexSim can only import basic shapes, so it is important to explode any compound objects in the AutoCAD drawing into their basic shapes. To explode compound objects in AutoCAD follow these steps:
    1. Select the objects you want to explode.
    2. Type "_explode" in the command prompt or select the explode command from the menus.
    3. Repeat the above steps until there are no objects that were exploded.

In FlexSim:

1. Drag a Background object into the model from the Library window under Visual.
2. Follow the steps in the Background Drawing Wizard.

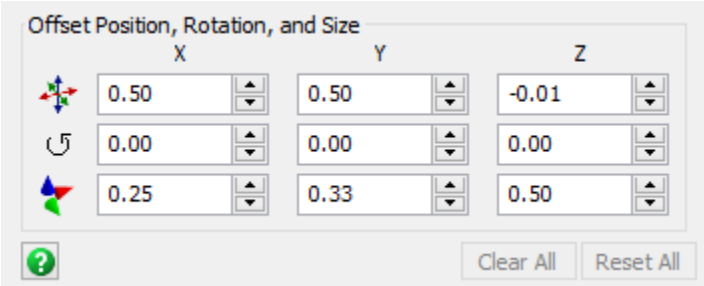
Shape Factors

Each media file that is imported has certain scaling and offset settings which may cause the 3D shape that you import to not fit within the object's boundaries. If this is case, you can edit the object's 3D shape factors to fit the 3D shape within the object's yellow bounding box.



The image above shows modified shape factors for a Processor's 3D shape. Notice that the yellow bounding box reflects the true position and size of the processor, but the 3D shape has been offset in the x direction.

Shape factors may be accessed through an object's General tab page.



: Change the position of the 3D shape within the bounding box.

: Change the rotation of the 3D shape within the bounding box.

: Change the size of the 3D shape within the bounding box.

Clear All / Reset All - When a shape frame is added to an object, it is given shape factors so that the shape will display inside the yellow box of the object. Clearing the shape factors from a shape frame will cause the shape frame to draw without regards to the object's yellow box, or the actual size of the 3D shape. Resetting the shape factors will reapply the originally calculated shape factors. By selecting the object's base frame, you can clear or reset all shape frame's shape factors. Or you can select an individual shape frame and clear/reset the shape factors for just that frame.

Shape Frames

You can animate an object or flowitem using 3D frames. This is done by creating multiple 3D shapes. Then within FlexSim you can tell an object which shape frame to display. This is useful if you want to change the look of an object like a Flowitem as it progresses through a model. An example of this might be a bottling line. The bottle starts out empty, then is filled and then a cap is placed on top. This could be displayed by using three different 3D shapes.

In versions of FlexSim prior to 7, shape frames had to be specified by the filename of a 3D shape. Now, shape frames are assigned manually by:

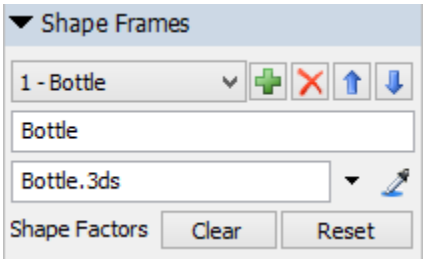
1. Adding a Shape Frame to the object
2. Setting a name for the frame
3. Specifying the 3D shape path

Once added, shape frames can be changed either by name, or index. See the Commands below.

Shape frames can be modified in two places:

- In the Quick Properties window under Shape Frames (see below).
- In the General tab page of an object's properties window.

Quick Properties



The shape frames section of the Quick Properties window does not always display for objects. For a FixedResource, TaskExecuter, or other non-flowitem object, the Shape Frames section will only show up if that object already has shape frames assigned to it through the General tab page.

Flowitems however, always have the Shape Frames section displaying. This is because it is far more common to change the shape of a Flowitem as it changes through the model run, than it is to modify a Processor or Operator.

Shape Factors - When a shape frame is added to an object, it is given shape factors so that the shape will display inside the yellow box of the object. Clearing the shape factors from a shape frame will cause the shape frame to draw without regards to the object's yellow box, or the actual size of the 3D shape. Resetting the shape factors will reapply the originally calculated shape factors.

Commands

To change an object's shape frame or to query an object to find its current shape frame, you'll need to use the following commands:

```
setframe(current, 1);
setframe(current, "Bottle2");
int curframe = getframe(current);
```

For convenience, object triggers like the OnReset, OnMessage, OnEntry/OnExit, OnLoad/OnUnload, etc. have a preset for changing an object's shape frame.

Level Of Detail (LOD)

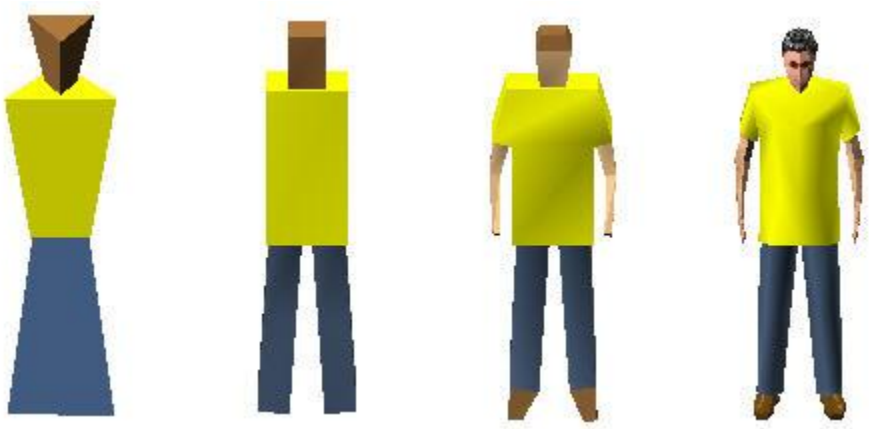
LOD files are 3D shapes that get progressively simpler and are tied together by a common name:

file.3ds / fileLOD1.3ds

If an object uses a 3D shape that has LOD then the 3D shape that the object displays is dependent on how far that object is from the view's camera position.

- 0 is the base LOD “file.3ds”
- 1 is the 2nd LOD “fileLOD1.3ds”
- 2 is the 3rd LOD “fileLOD2.3ds”
- N is the nth LOD “fileLODn.3ds”

LOD Files are used to show higher level objects when the viewer is close up and lower level objects when the viewer is farther away. LOD improves the speed at which a model will display in the view window. LOD allows for the use of more polygon intensive 3D shapes by displaying only a few high polygon count shapes at any given time.



Preparing LOD Files

To prepare a group of 3D Shapes for use as an LOD file do the following:

1. Gather together two or more shapes that you want to use as LODs of the same object.
2. Get the LOD files saved as the same file type (3ds, wrl).
3. Make sure that each LOD uses or at least isn't disfigured by the base file's .xds file (each frame can have it's own .yds file).
4. Name the LOD files appropriately. The first file is the “name.3ds” or “name.wrl”. Each successive file is “nameLOD#.3ds” or “nameLOD#.wrl”. For example, Queue.3ds, QueueLOD1.3ds, QueueLOD2.3ds, etc.

To import an LOD shape just follow the steps used in the section on Importing 3D Media. Use the base object for the name of the file that you select.

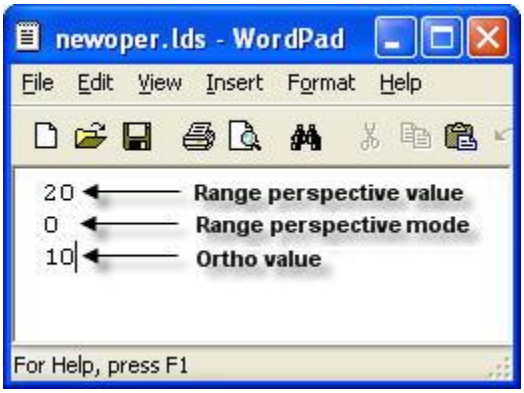
LDS/LRL Files

lds / lrl files are used to determine the distances at which the different LOD's will be shown. lds / lrl files must have the same names as the objects they modify:

- crane.3ds – crane.lds
- crane.wrl – crane.lrl

lds / lrl file info

The lrl/lds file is a text file made up of three numbers separated by new lines: The range perspective value, the range perspective mode, and the range ortho value.



- The range perspective value is the distance value at which the LOD files will change in the perspective window.
- The range perspective mode value is either 1 or 0. 0 means linear (slightly faster). 1 means inverse distance (more coverage at farther distances).
- The range ortho value is the distance value at which the LOD files will change in the ortho window.

Miscellaneous Concepts

1. Advanced Undo
2. Custom Libraries
3. FlexSim Tree Structure
4. FlexSim XML
5. GUIs
    - GUI Events and View Attributes
    - View Attributes Reference
6. Kinematics
7. License Activation
8. Sampler
9. SQL Queries
10. State List
11. Web Server
12. When to Compile

Advanced Undo

1. Concepts
2. Example

Advanced Undo Concepts

This topic describes advanced mechanisms for storing undo history. This is meant mostly for developers of custom user interfaces and libraries.

Topics

- **When Advanced Undo is Needed**
- **Creating an Undo Record Explicitly**
- **Ignoring Undo**
- **Undo Tracking on a Custom Object's Drag**

When Advanced Undo is Needed

In FlexSim, for the most part, undo/redo just works. You don't need to worry much about what is being recorded to the undo history, because it's all automatic. However, there are some instances where you might need to customize the undo history recording to suit your needs. Some examples of when you might want to customize undo history recording are:

- When you have created a customized object, and that object changes certain variable values as the object is dragged around in the model, i.e. in its OnPreDraw or OnDrag functionality. You may want those values to go back to their original state before the object was dragged.
- If you have custom functionality in a user library's drop script that you do NOT want recorded in the undo history. By default, whenever the user does a drag/drop operation from the library icon grid, FlexSim will automatically start an "aggregated undo" operation, which essentially retools several low-level commands to record changes that are made. We'll talk about this in more detail later, but for some dropscripts, you may need to temporarily "turn off" undo recording, then execute your functionality, then turn it back on. For example if you are performing functionality that opens a window.
- If you create a tool window that is used to modify the model, like the Edit Selected Objects tool windows, then you'll need to add code to the buttons of those windows that explicitly does undo recording. In FlexSim, all things that are done in a 3D view, a tree view, or a table view, as well as all operations done in tool windows that focus around those views, are undo-able, so if you design a tool window that is used with one of these views, you'll need to make it undo-capable.

Creating an Undo Record Explicitly

To create an undo record before executing a piece of code, call beginaggregatedundo(). Pass in the view that this undo is associated with. If you pass in NULL as the view, then it will be associated with FlexSim's global undo history (MAIN:/project/undo/). Pass in a description of the operation being done ("ChangeColor"). The beginaggregatedundo() command will return back a unique id for the undo record that you've created. Then execute the code that you want to be undo-able, then call endaggregatedundo(), passing in the view and the undo id.

When you call beginaggregatedundo(), FlexSim will retool several low-level commands, so that subsequent calls that you make to those commands, or calls to those commands by other commands that you call, will record changes to the undo history. The commands that are retooled are as follows:

- nodeadddata()
- setnodenum()/set()
- setnodestr()/sets()
- nodedeldata()
- nodeinsertinto()
- nodeinsertafter()
- nodejoin()
- nodepoint()
- nodebreak()

- setname()
- clearcontents()
- createcopy()
- createinstance()
- switch_cppfunc(),switch_flexscript()... (all switch_... commands)
- transfernode()
- transfernodeobj()
- moveobject()
- setrank()
- destroyobject()
- destroynode()

Ignoring Undo

If you are inside of an aggregated undo and you want FlexSim to ignore a set of commands, call beginignoreundo(), execute the commands, then call endignoreundo(). This is especially useful in dropscripts of a user library. FlexSim automatically creates an aggregated undo record that applies to all functionality executed as part a drag/drop operation from the library icon grid, so all code that executes in a drop script is being recorded as part of an undo record. To bypass this for certain functionality, surround it with beginignoreundo() and endignoreundo()

Undo Tracking on a Custom Object's Drag

You may also want FlexSim to track changes to a custom object's variable value, or the location, rotation and scale of an object that the custom object changes as it is dragged. One example of this being used is in the Crane object in FlexSim's standard library. When you drag on one of the Crane's sizers, there are actually several objects behind the scenes that the Crane resizes as you drag. Also, the Crane has three variable values defining the x,y, and z location of the Crane's frame that change as you drag the Crane to a different position in the model. In order for undo functionality to work on the Crane, it must tell FlexSim to track undo information on those objects that it is changing as the user drags.

To add undo tracking to an object or number variable, call applicationcommand("addundotracking", tonum(view), tonum(obj_or_variable)). The Crane performs this as part of its OnClick event, when the user clicks on the Crane.

Advanced Undo Example

The following are snippets of example code using custom undo.

Simple Undo

```
int undoId = beginaggregatedundo(c, "Set Object Rank");

setrank(anObj, 5);

endaggregatedundo(c, undoId);
```

Undo with an Ignore

```
int undoId = beginaggregatedundo(c, "Change Object Shape");

setnodestr(shape(anObj), shapePath);

int index = getshapeindex(gets(shape(anObj)));

beginignoreundo();

if (getshapeindex(shapePath)==0) {

     autoloadallmedia(anObj);

     index = getshapeindex(gets(shape(anObj)));

}

endignoreundo();

set(shapeindex(anObj), index);

applyshapefactors(anObj);

endaggregatedundo(c, undoId);
```

Custom Libraries

1. Concepts
2. ModelLibraries Node
3. Example

Custom Libraries Concepts

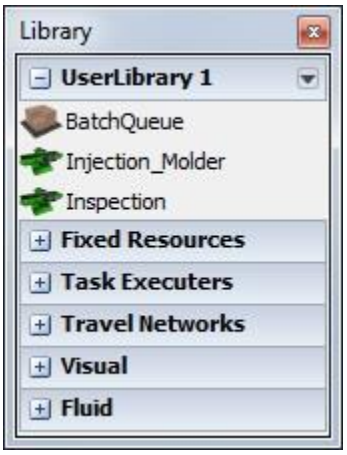
Topics

- **Overview**
- **Adding objects to a custom library**
- **Editing Library Objects**
- **Saving / Removing Libraries**
- **Dropscript and Droppath**
- **Automatic Install**
- **Customizing the Library Icon Grid**

Overview

FlexSim lets you create and configure special libraries in addition to the standard library set. These are referred to as user libraries. You can create custom defined functionality on objects, and then add those objects to a library for use in other parts of your model or in other models. You can save these libraries and then load them into other projects later on. You can also define a set of objects in the library to be automatically installed to your model when a new model is created or when you load the library.

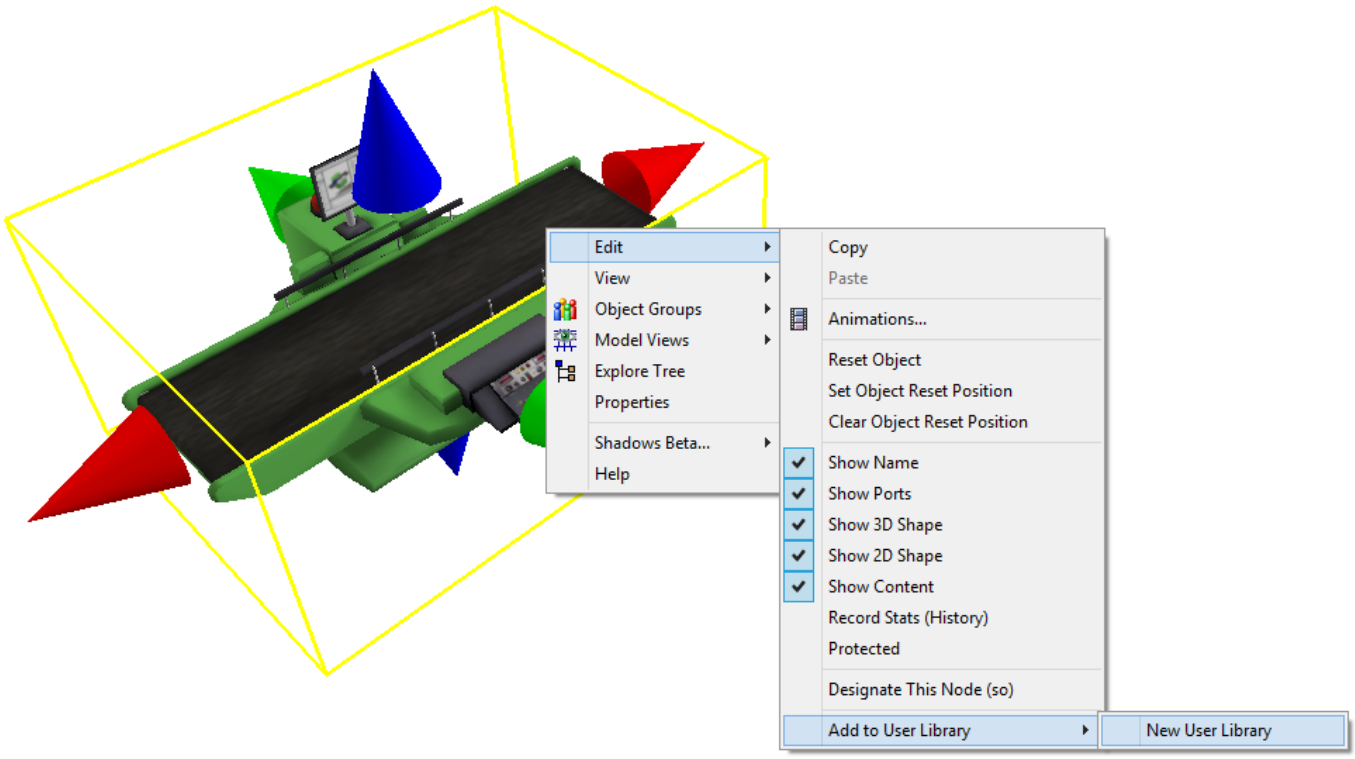
You can create new User Libraries and Open User Libraries from the File Menu.



The custom library, or user library, mechanism provides you with flexibility in many areas. The most widely used functionality of the user libraries is to reuse customized objects in a model, but you are not confined to only that functionality. User libraries allow you to automatically install custom objects and data into models, install objects and data into the main project and view tree, and execute code when an object in the icon grid is dropped into the model. All of this functionality stems from two mechanisms within the user library functionality. First is the droppath and dropscript mechanism, which allow you to customize what happens when an object is dropped into that model. Second is the automatic install mechanism, which allows you to install objects or data when the user does operations like creating a new model or opening a model. This topic will discuss these two mechanisms in detail.

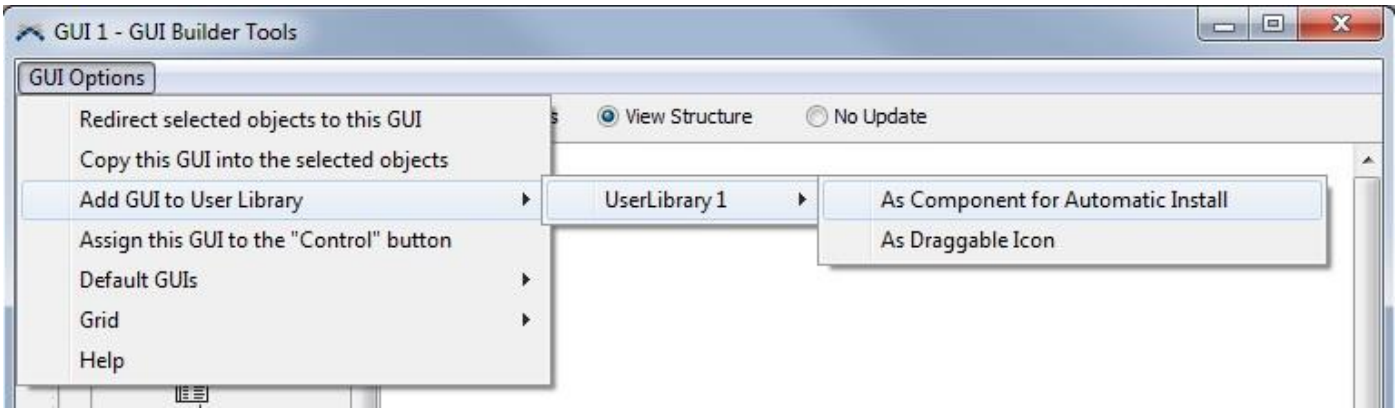
Adding objects to a custom library

There are several types of objects that you can add to a user library. You will most commonly add a standard object like a Processor or Queue whose properties you have modified to fit a custom modeling situation. You could also start from scratch with a BasicFR or BasicTE object, implement your custom behavior, and then add it to a library. Another possibility is to use a VisualTool as a container of a sub-model, and add the whole sub-model to the library. To add these types of objects to a user library, right click on the object in the 3D model view and select **Edit > Add to User Library**.



If you already have User Libraries loaded, a list of those libraries will appear in the right click menu.

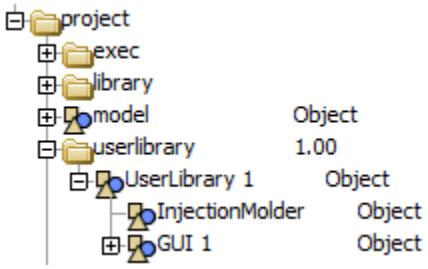
You can also add GUIs, Global Tables, Flowitems, and User Commands to a custom library. You can access this capability from the windows in which you edit these respective objects. The figure below shows a menu option in the GUI editor to add the GUI to a library.



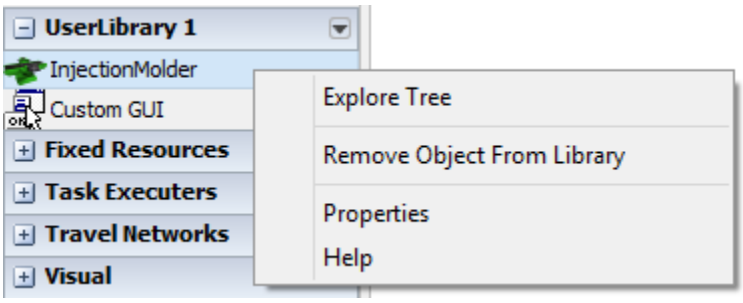
Editing Library Objects

When an object is added to a library, a copy of the object is created and put in the library. This means that once it is added in the library it has no more linkage to the original object. It is a copy and can be changed on its own, separate from the original object.

The User Library is stored in the Main Tree in the userlibrary node.

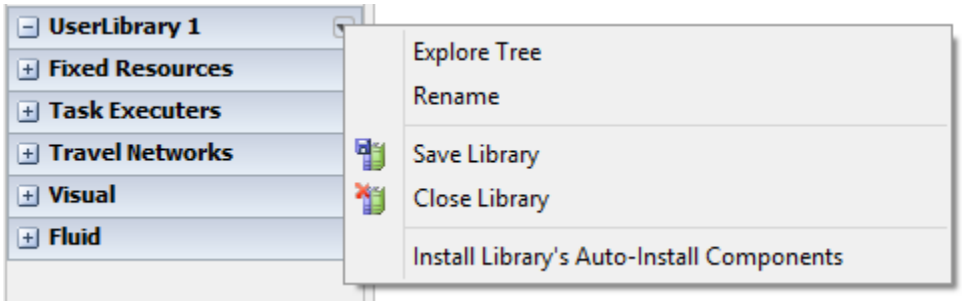


You can also access the properties of your library object through the right click menu on the Library Icon Grid.



Saving / Removing Libraries

To Save or Remove a User Library, click the arrow next to the User Library in the Library Icon Grid.

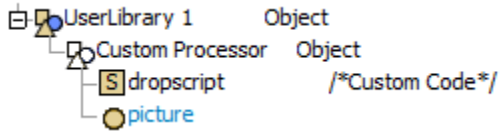


Dropscript and Droppath

The droppath and dropscript mechanism allows you to customize what happens when an object is dropped into the model. This is done by creating a custom object in the user library, and then adding either a "dropscript" or a "droppath" attribute to the object and specifying the data for the dropscript or droppath.

Dropscript

If a dropscript attribute is added, then the dropscript node will be executed as flexscript, and values will be passed into the flexscript function that give it information about the drop.



Note: In order for an object to display in the Library Icon Grid, it needs to have a picture attribute as shown above in our Custom object.

There are 5 access variables you can get within the dropscript function. They are:

parnode(1) - the object that was dropped onto. If the user dragged onto a blank area in the model, then this variable will be 0, or NULL. In such a case, you can usually assume that the user intends to drop it into the model. However, this is not always the case. For example, the user may drop the object into a 3D view that is viewing a subspace of the model, like a VisualTool. To ensure a safe drop in this case, you should use the viewfocus of the view as the drop point: node(">viewfocus+", parnode(5)).

parval(2), parval(3), parval(4) - the x, y and z location of the drop, respectively. If the parnode(1) is NULL, then this is the location within the model space (or the view's viewfocus' space). If parnode(1) is not NULL, then this is the location within parnode(1).

parnode(5) - the view window onto which the object was dropped. Use node(">viewfocus+", parnode(5)) to get access to the viewfocus of the view.

If within the dropscript function you would like to do a standard object drop, as if it were a regular user library object being dropped, then you can use **dropuserlibraryobject** command. This command executes the same functionality as when a regular object is dropped from a user library object. The first parameter is

the object to drop, and the next five parameters are the onto object, x, y, z, and onto view, just like the dropscript function itself. The command returns a reference to the object that was created.

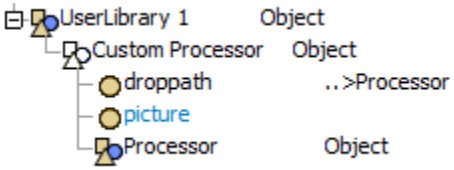
Return Value - Your dropscript function should also return a reference to the object you created. This allows FlexSim to do some extra work like checking to see if there were any c++ nodes created, and if so, set a flag to notify you that you need to compile the next time you run the model.

The dropscript mechanism allows you ultimate flexibility with user libraries. You can use it in the standard way, for example to do a regular object drop but also execute some extra code to initialize the dropped object. You can also use it in more non-standard ways. Some ideas may be to use a dropscript to add a curved or straight section to a conveyor belt, or to add levels or bays to a rack object. You could also use a dropscript to add code to triggers of an object, or to set certain parameters of the object. You could add collision spheres to an object, or add a set of functionality to the model's tools folder. There are any number of possibilities.

Refer to the Example page for examples on using the dropscript node.

Droppath

The droppath mechanism is a method of indirection. It is a way that you can refer to another object that you want to be dropped when the icon is dropped into the model. This is usually used with automatic install functionality, but it can also be used with an icon in the icon grid. The method of creating a droppath object is the same as for creating a dropscript. You add an object to the library, and give it object data, but this time you give it an attribute named "droppath". You should not toggle the node as flexscript, but should give text data to the droppath attribute and specify in the text a path to another object that should be dropped.

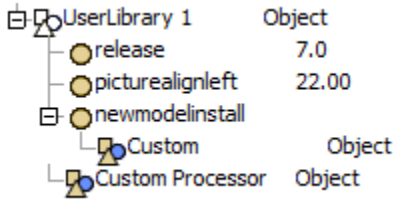


For example, you may give the droppath attribute the text: MAIN:/project/library/NetworkNode. This will effectively cause a network node to be created when you drag your object's icon into the model. You might ask why not just add a network node to the user library, or why not just make them go to the regular library and drop a network node from there. To the former question, if you plan on using this user library for a long time, you may want the network node to be compatible with future versions of FlexSim. You may want to drop whatever is the latest version of the network node, without having to update the user library with each new version. Also, why have redundant data in your user library if all of it is already there in the regular library? To the latter question, it may be useful to have the network node as a droppable icon in your library if the user uses the object often. It reduces the number of mouse clicks needed.

The droppath can either be an absolute path *MAIN:/project/library/fixedresources/Processor* or relative *..>Processor*.

Automatic Install

The automatic install mechanism allows the user library to install objects and functionality to the model when the user performs certain actions like creating a new model, or opening a model. Automatic install is available through FlexSim's user interface for GUIs, Global Tables, Flowitems, and User Commands.



To create a custom automatic install, you can add special nodes to the user library object. The specific set of valid node names are listed as follows:

newmodelinstall - This node is installed when the user creates a new model. It is also installed when the library is loaded.

startupinstall - This node is installed when the library is loaded if the user has designated this library as a library to load during startup.

loadinstall - This node is installed when the library is loaded explicitly by the user.

openmodelinstall - This node is installed when the user opens an existing model. You may use this node to check if the model contains the library's components, and if not, installs them. You could also use it to update components in the model if those components are from an earlier version of the user library.

Customizing the Library Icon Grid

There are a few attributes that can be added to objects in your custom library that will affect they way they are displayed by the Library Icon Grid.

Note: Some visual attributes will not be update in the library icon grid without closing and reopening the view.

Pictures

The picture attribute specifies the path to an image to display in the icon grid. Leaving the path blank will display the object name only. To give your object a Processor image, your picture attribute would contain the path:

bitmaps\processorpicturesmall.png

You can also specify the left alignment by changing the picturealignleft node value.

Icon Grid Width and Height

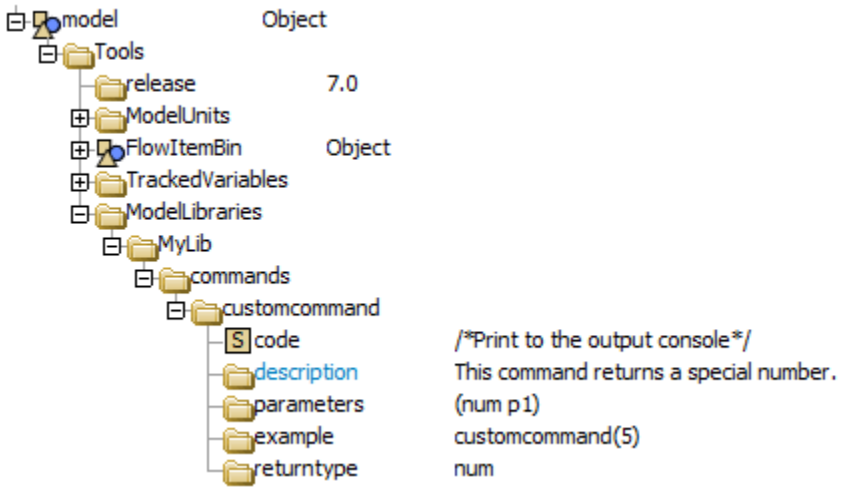
You can designate the size of icons in the library icon grid. Just give the library attributes named "cellwidth" and "cellheight". Each attribute should contain a number that is the width and height of one icon in the icon grid in pixels.

ModelLibraries Node

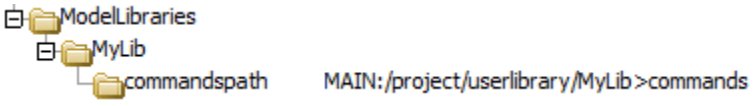
You can add many facets to a model, such as user commands, global variables, global macros, etc. where the definition is "hidden" from the user, so that they don't see it in their User Commands window, Global Variables window, etc. These can also simply refer back to structures in your library, so that you don't have to continually copy stuff into the model when updating. This is done by adding a node named "ModelLibraries" to the model's Tools folder and giving its substructure various data. While you don't need to have a custom library loaded to have this feature be part of a model, it will probably be most used for user libraries, hence we include this information with this topic.

To create this functionality, add a node into model/Tools, and give it the name "ModelLibraries". Inside of that node, add another node, and give it the name of your library, i.e. MyLib. Inside of that node, there are several nodes that can be added. Their name defines their meaning to FlexSim. Names and meanings are listed below:

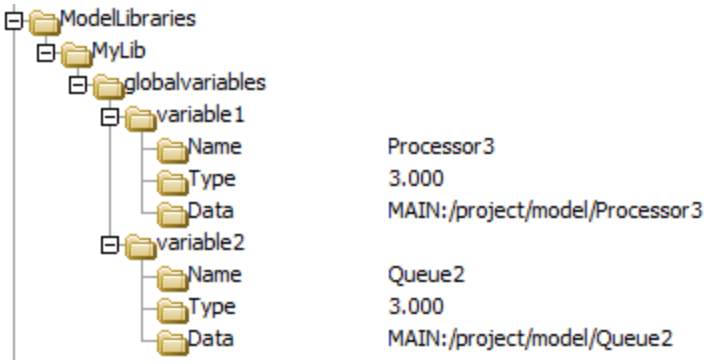
- **commands** - If you add a node named "commands", you can place commands inside that node. These commands will be accessible to the user to call, and will be documented in the commands documentation, but won't be visible in the User Commands window. The structure of each command should be the exact same as the structure of model/Tools/UserCommands.



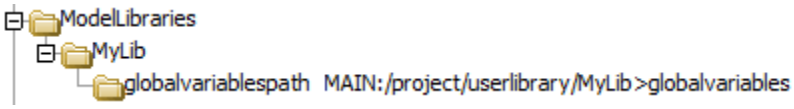
- **commandspath** - You can add a node named "commandspath" and give it text data that specifies a path to an alternate location that holds the definition of the commands. In this case the structure of the path's destination should be the exact same as the structure of model/Tools/UserCommands. This allows you to leave your command definitions in the user library, and then just have this node refer to it.



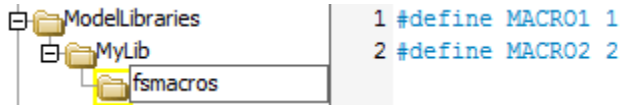
- **globalvariables** - You can add a node named "globalvariables" and give it the same sub-structure as found in Tools/GlobalVariables to define additional global variables that are available to the end user but "hidden" from the "Global Variables" window.



- **globalvariablespath** - Like commandspath, you can redirect the location of where global variables are defined

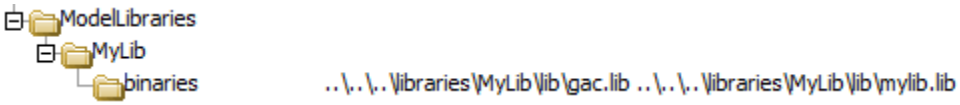


- **fsmacros** - Here you can add a node named "fsmacros", give it text data with multiple #define macros. These macros will be visible (blue) to the user and will be added as options in auto-completion hints.



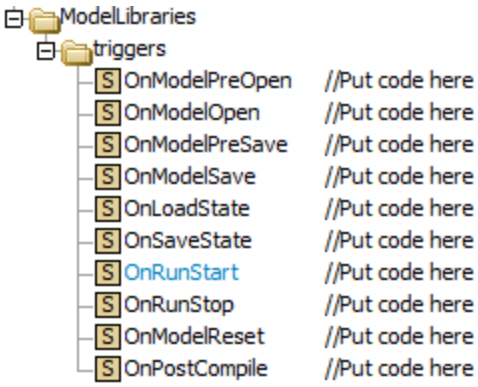
- **fsmacrospath** - Like commandspath, you can redirect the location of where flexscript global macros are defined with the "fsmacrospath" option.

- **binaries** - You can add a node named "binaries" and give it text data that specifies a list of additional .lib or .obj files that you want to link to when FlexSim is compiled. This is only needed if you have c++ toggled code that must be compiled, or if you expect your end users to define c++ code that can access pre-compiled functionality. This text will be added to the "Additional Library Dependency" field during the C++ linker phase. The linker's library path is specified as \program\system\lib in FlexSim's install directory, so to link with a lib file in FlexSim's libraries directory, you would specify a path like: ..\..\..\libraries\MyLib...



- **binariespath** - Like commandspath, you can redirect the location of where linker options are specified with the "binariespath" option.

- **triggers** - You can also add several triggers to fire at certain points. To do this add a node and give it the name "triggers", and add various nodes into that triggers node.



- Valid triggers are:
  - **OnModelPreOpen** - Fired immediately after a model tree is loaded, and before any model initialization code is fired.
  - **OnModelOpen** - Fired after a model has been opened and all initialization code has finished.
  - **OnModelPreSave** - Fired just before a model is saved to file.
  - **OnModelSave** - Fired just after a model has been saved to file.
  - **OnLoadState** - Fired when a state load is performed.
  - **OnSaveState** - Fired when a state save is performed.
  - **OnRunStart** - Fired whenever the user presses the "Run" button, as well as whenever the go() command is called.
  - **OnRunStop** - Fired whenever the user presses the "Stop" button, as well as whenever the stop() command is called.
  - **OnModelReset** - Fired whenever the user presses the "Reset" button, as well as whenever resetmodel() is called.
  - **OnPostCompile** - Fired after the user compiles the model.

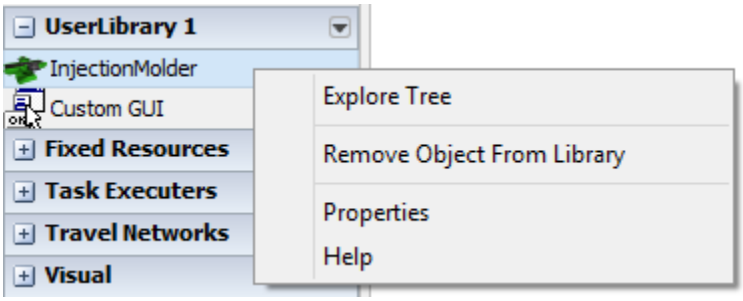
Custom Libraries Example

Topics

- **Dropscript Example**
- **Dropscript Dynamic Parameters Example**
- **Automatic Install Example**

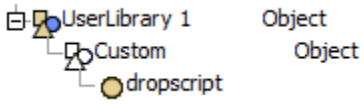
Dropscript Example

Create a new user library by going to File > New User Library. Click on the down arrow next to the newly created User Library in the Librar Icon Grid and select Explore Tree.



You should now see an empty tree.

Insert a new object into the library by right clicking on it and selecting Node > Insert Into, or by left-clicking on it and hitting the Enter key. Expand the user library tree so you can see the node you created by pushing the plus button. Give it the name "Custom". Now add object data to the node by right-clicking on it and selecting Node > Add Object Data or by left-clicking on the node and hitting the O key. Click on the > button to expand the object. Then insert an attribute node by clicking on the object node and hitting the Enter key. Give the attribute node the name "dropscript". The tree should appears as follows.

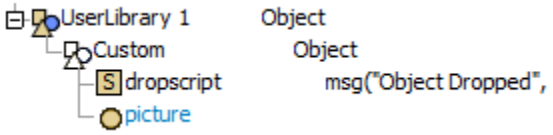


Add text to the dropscript attribute by right-clicking on it and selecting Node > Add Text Data or by left-clicking the node and pressing the T key. Toggle the node as flexscript by right-clicking on it and selecting Build>Toggle as FlexScript. Give the attribute the following flexscript text:

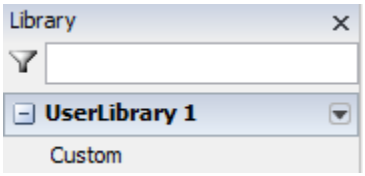
```
msg("Object Dropped",
  concat(
    "Onto Object: ",
    getname(parnode(1)), "\n",
    "X: ",
    numtostring(parval(2),2,2), "\n",
    "Y: ",
    numtostring(parval(3),2,2), "\n",
    "Z: ",
    numtostring(parval(4),2,2), "\n",
    "Onto View: ", getname(parnode(5))
  )
);
```

Click off of the attribute node, then right-click it and select Build>Build Node FlexScript.

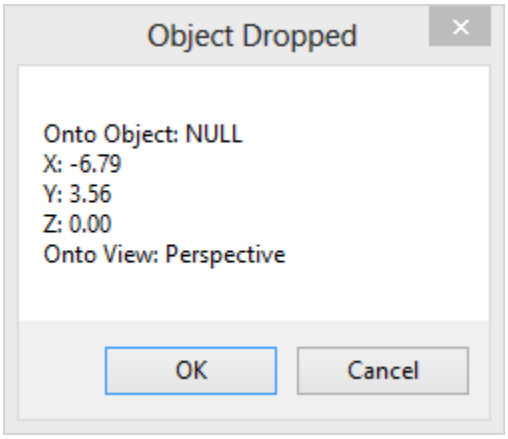
Finally, add a picture attribute to the attribute tree of the object by right-clicking on the drop-script node and selecting Node > Insert After. Name the new node "picture". The libraries icon grid will only show the object in the grid if the object has a picture attribute. Usually this is a path to a bitmap file that represents the picture that is shown in the icon grid for that object. In our case we won't worry about that and will just leave the picture attribute blank, which causes only the object text to be shown in the grid. The tree should appear as follows.



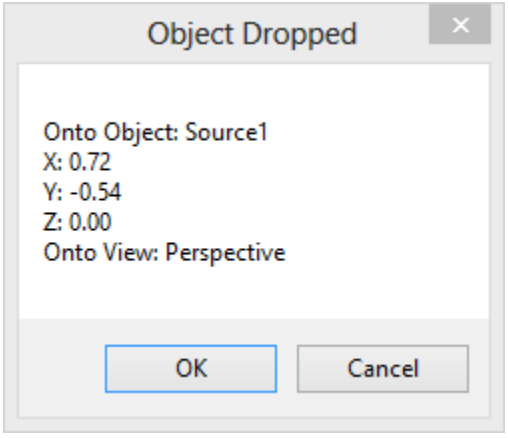
The library icon grid should appear as follows.



Now drag the Custom icon from the Libary Icon Grid into the model. A message should appear.



This message is the code of the dropscript being executed. It shows the object that you dropped onto if there is one, an x, y and z location of the drop, and the view window that the object was dropped into. You can also drop it onto another object. Drag a regular library object into your model, then go back to the user library and drag the Custom object onto the object in the model. The following message should appear.



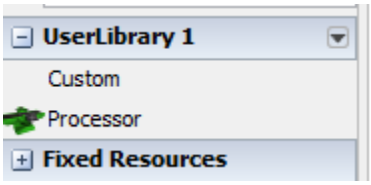
This time the message shows that it was dropped onto Source1. The x, y and z locations are now relative to the source object.

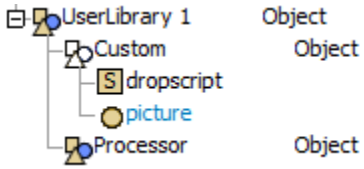
Dropscript Dynamic Parameters Example

In this example, we will have an object that is created through our User Library that will have it's size defined by a Global Table.

Create a new User Library as described in the above Dropscript Example.

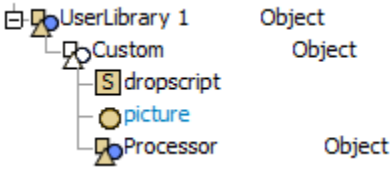
Drag-and-drop a Processor object from the Library Icon Grid to the 3D view. Right click the Processor and select Edit > Add to User Library > UserLibrary 1 to add it to our library.



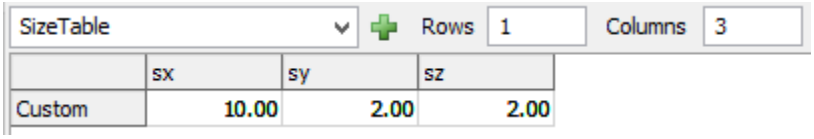


We want to dynamically change the size of our Processor as it is dragged into the 3D view. In order to accomplish this, we will use our Custom object with code in our dropscript node. We don't want our Processor to show up in the Library Icon Grid, so we'll copy it into our Custom object as an attribute node.

Copy the Processor object (left click and hit Ctrl+C). Left click on the Custom object and open the object's attribute tree if it is not currently open by pressing the . Hit the enter key to create a new attribute node in the Custom object. Left click the new node and hit Ctrl+P to paste the Processor object. You can then delete the original Processor object.



Create a new GlobalTable through Tools > Global Tables > Add. Name the table SizeTable. Make your GlobalTable look as follows:



The values from this table will define the size of our processor.

Back in the Custom object, update the dropscript code to the following:

```
treenode ontoObj = parnode(1);
double x = parval(2);
double y = parval(3);
double z = parval(4);
treenode ontoView = parnode(5);
treenode newProcessor = dropuserlibraryobject(node("..>Processor", c), ontoObj, x, y,
z, ontoView);
setnodenum(spatialsx(newProcessor), gettablenum("SizeTable", 1, 1));
setnodenum(spatialsy(newProcessor), gettablenum("SizeTable", 1, 2));
setnodenum(spatialsz(newProcessor), gettablenum("SizeTable", 1, 3));
return newProcessor;
```

Right click the dropscript node and select Build > Build Node FlexScript.

Now create a Custom object by dragging and dropping it from the Library Icon Grid to a 3D view. Notice the size of the object has been altered by our dropscript.

Note: This is the proper way to dynamically change parameters or variables of a custom library object when it is dropped into a view (though it does not matter where the created object is contained). If you add a dropscript node straight to the Processor, the Processor will not be created. If you try and create the object in the Processor's dropscript node, you will cause FlexSim to get into an infinite loop and crash.

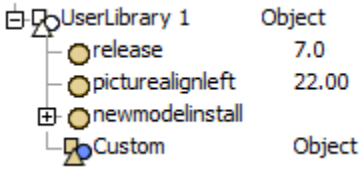
Automatic Install Example

For this example, we will have the Processor object from the Dropscript Dynamic Parameters Example be automatically installed when the user creates a new model.

Open the GlobalTable SizeTable. Click on the button and select **Add to UserLibrary 1 > As Auto-Install Component**.

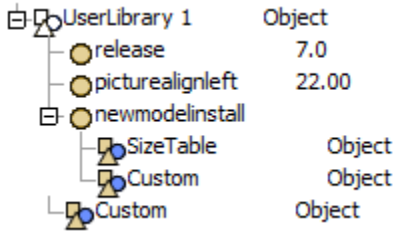
Open UserLibrary 1's Tree view by click the arrow next to the User Library in the Library Icon Grid and select Explore Tree.

Expand the library's attributes tree by pressing the button next to UserLibrary 1. The tree should appear as follows.



Press the plus button next to the newmodelinstall to expand the node. Notice our SizeTable already appears inside the node. This is important as our Custom object's dropscript refers to this GlobalTable.

Create a subnode of newmodelinstall by left-clicking the node and hitting the Enter key. Copy and paste the Custom object onto this new node.



Now you can hit the "New" button on the toolbar of File > New Model. You'll notice that there is a Processor1 object in the 3D view and our SizeTable in the GlobalTables. The newmodelinstall folder acts a set of objects that will be "dropped" into the model when a new model is created. The object's are "dropped" in that the same functionality is executed as when you actually drag an object from the icon grid. In the case of a dropscript object, the script is executed as if the object were dropped at the point (0,0,0) in the model.

FlexSim Tree Structure

FlexSim is completely designed around the concept of a tree structure. All information in FlexSim is contained within the FlexSim tree, including the library objects, commands, and all model information. This tree hierarchy is made of individual nodes that link together and hold information.

Nodes

A node is the building block of a FlexSim tree. All nodes have a text containing the name of the node. Nodes can simply be a container for other nodes, can be a keyword used to define an attribute of an object, or may have a data item.

The data item types which may be attached to a node are: number, string, object, or pointer. To attach data to a node, right-click on the node and go to the Insert menu option. You will see the four options to add data to a node. There are also shortcut keys for adding number, string(text), object, or pointer data. These are the keys N, T, O, and P. To add data to a node using a shortcut key, click on the node, then press the appropriate key. Nodes can also hold executable code. To make a node executable, first add string data to the node, and then toggle the node as either a C++ or a FlexScript node. To toggle a node as one of these types, right-click on the node and go to the Build menu.

The symbols for the different types of nodes are shown here:

Standard:

Object:

Attribute/Variable:

Function (FlexScript):

Function (FlexScript, not built):

Function (C++):

DLL Linked Function:

Global C++ Function:

Simple Data:

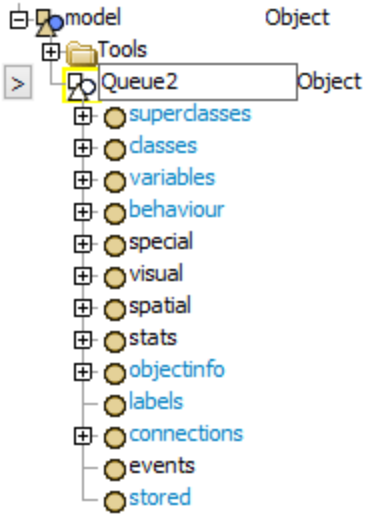
Nodes can be added and deleted from the tree. To delete a node, simply click the node and then hit the delete key. To insert a node, right-click on an existing node and choose Edit > Insert. This will add a new node immediately after the node that was clicked on. The shortcut for this operation is to hit the spacebar after first highlighting a node.

Nodes can also contain a sub list of nodes called the content branch. If a node contains sub nodes it can be expanded by pressing the ⊞button. To insert a node into the content of an existing node, choose the option Edit > Insert Into, or hit the Enter key as a shortcut.

A node that has object data may contain a second sub list of nodes that are contained in a separate branch of the tree. This sub list of nodes is called the object attribute tree, and contains data that describes the properties of the object. A node containing object data may typically be referred to as an object node.

When you click on an object node you will see a greater than symbol to the left of the node. Clicking on this button will open the object attribute tree branch.

The following picture shows an expanded object attribute tree for the Queue object in the library tree.



For nodes with object data, the attribute tree can contain many special attribute nodes. If a node is inside an object and has the name of a key attribute, it will have a special meaning to the object. The actual meaning of the attribute depends on what the attribute is and the object type. As an example, there are attributes for an object's position: 'spatialx', 'spatialy', 'spatialz'. The list of available attributes in FlexSim is found in attribute hints.

In addition to containing all model, library, and project information, the FlexSim tree also stores all windowing and interface information. All open windows, menus, toolbars and buttons have a corresponding representation in the FlexSim tree. We call these types of nodes view objects.

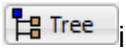
General Organization Trees

FlexSim's root tree structure is split up into two parts. These are the Main Tree and the View Tree.

The main tree contains the Executive data, the Library, and the Model.

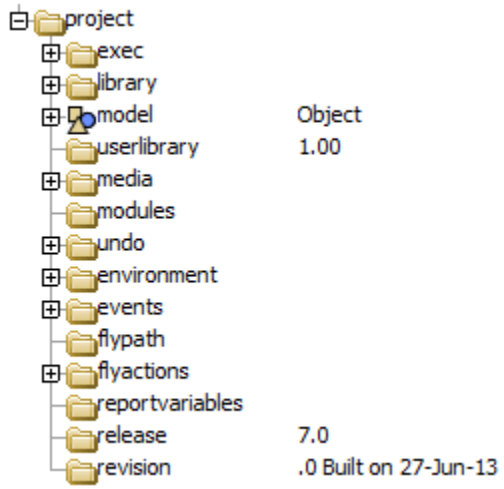
The view tree contains information on windows, editors, and other user interfaces. It also manages active windows.

Main Tree

To view the main tree, click on the 

in the FlexSim toolbar, or select the main menu option: View > Model Tree. A new tree window will appear. In the Quick Properties window you'll see a Tree Navigation section. Click on the **Main**.

The main tree holds many of the higher level functions in FlexSim. It also includes the following crucial sub trees: exec, library, model, undo, media.



Exec

This tree contains simulation executive data. This includes the simulation time, the eventlist, as well as other information with running a model.

Library

The library of objects used by the model.

Model

The simulation model.

userlibrary

All loaded custom user libraries.

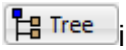
Media

Stores Images, 3D models and Sounds.

Undo

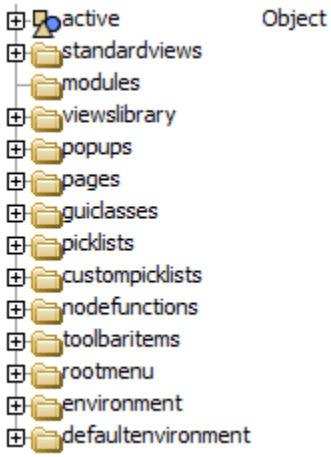
Holds undo history. A numerical value for this node is the limit on the number of undo steps. If there is no numerical data, undo will be disabled. Undo functionality may also be globally disabled.

View Tree

To view the view tree, click on the 

in the FlexSim toolbar, or select the main menu option: View > Model Tree. A new tree window will appear. In the Quick Properties window you'll see a Tree Navigation section. Click on the **View**.

The view tree contains data for creating, storing, and using graphical user interfaces for objects.



Active

This stores all of the currently open windows for the interface.

Standardviews

All of the non-property windows are stored here.

Popups

Popups are used throughout FlexSim, mostly for trigger and picklist options as well as utility purposes.

Pages

Stores all of the Object Property Windows.

Picklists

Preset code for picklist options.

Keyboard Shortcuts

Spacebar - Insert a new node after.

Enter - Insert a new node into.

N - Add number data to the highlighted node.

T - Add string (text) data to the highlighted node.

O - Add object data to the highlighted node.

P - Add pointer data to the highlighted node.

FlexSim XML

FlexSim saves it's models, library and tree files in XML format. There are many advantages to using this capability in your model development, including:

- Since XML is an ascii/text based format, it increases the utility of using content management and versioning software such as Subversion, CVS, Microsoft Visual SourceSafe, git, etc. to manage the development of a model. These systems automatically track a history of changes to a model or project, and for ascii/text based files, they allow you to see line-by-line change logs for the files, as well as revert line-item changes if needed.
- With the added benefit of versioning systems comes the ability to develop a model concurrently by different modellers using a much more stream-lined method. No more saving off individual t files from one modeller's changes to a model and loading them manually into the master model. When saving to XML format, if one modeller is working on one portion of the model, only that portion of the model file changes when saved, so when the modeller checks his changes into the versioning system's repository, another modeller can automatically merge those changes into his copy. If there are conflicts, where two modellers change the same part of the model, then the versioning system has tools that allow modellers to easily see in the XML file where the conflicts occur and quickly perform the manual conflict resolution.
- FlexSim also has the added ability to distribute a single model across multiple XML files. This increases your control over how to manage the model development in your versioning system, and makes conflict resolution even easier.
- The distributed save mechanism also opens the door for much more automated control of FlexSim. By saving small pieces of your model off into separate XML files, you can auto-regenerate one or more of those XML files outside of FlexSim, consequently changing the configuration of the model for the purpose of running different scenarios.

Tutorial

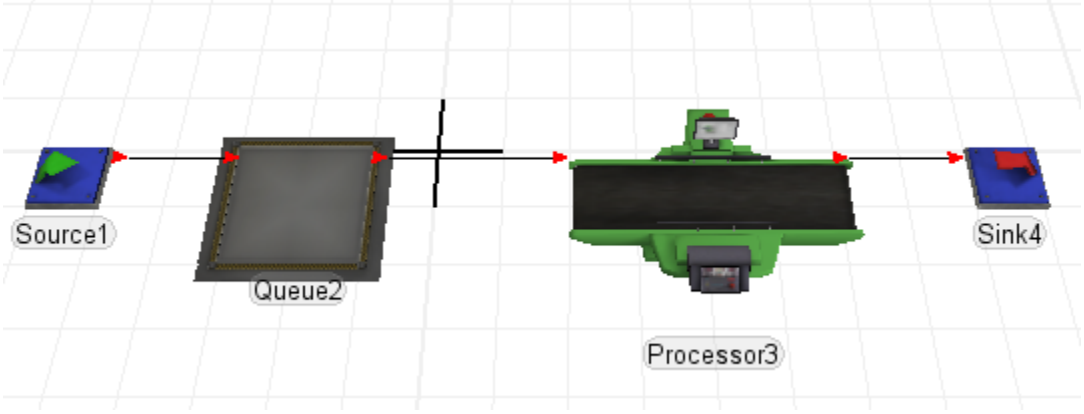
This tutorial will take you through saving a model as XML, and how you can configure the distributed save.

Topics

- **Build a Simple Model**
- **Save in XML Format**
- **Version Management Utilities**
- **Distributed Save**
- **Why Distribute?**

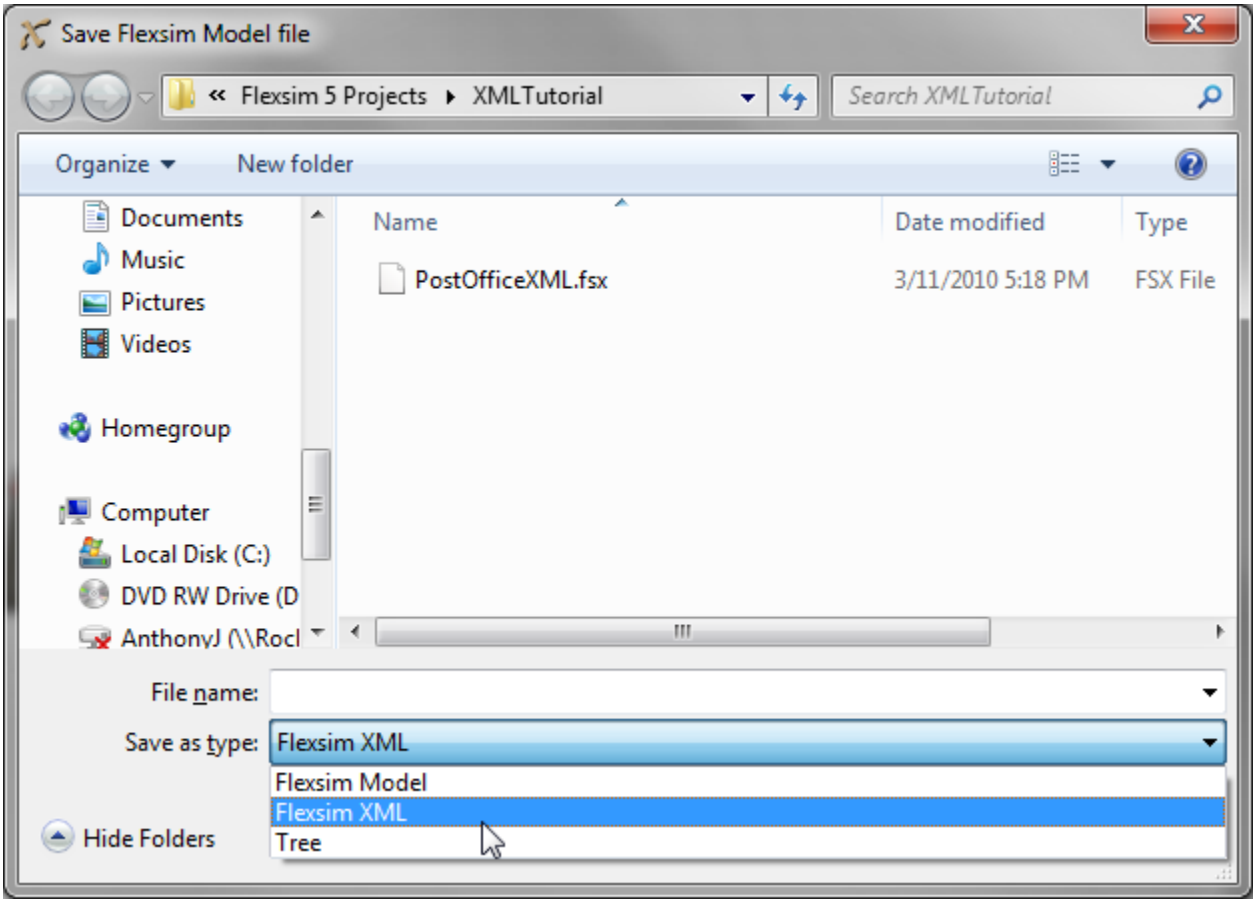
Build a Simple Model

First build a simple example model containing a Source, Queue, Processor and Sink.

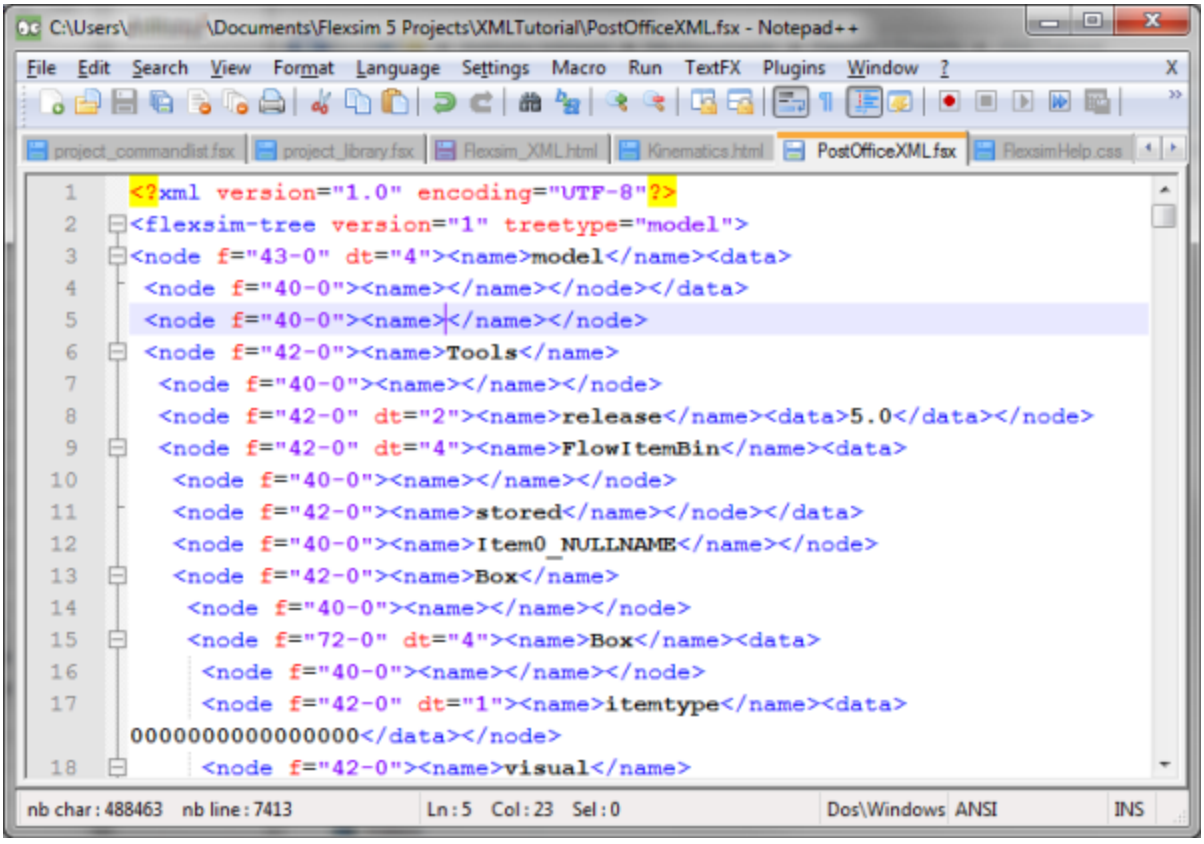


Save in XML Format

Save the model, and in the save dialog, choose "FlexSim XML" from the drop-down at the bottom. Save the model as "PostOfficeXML.fsx".



Now you've saved the file in FlexSim's XML format. You can view the file in a regular text editor like Notepad, Visual Studio, Notepad++, EditPlus, etc.

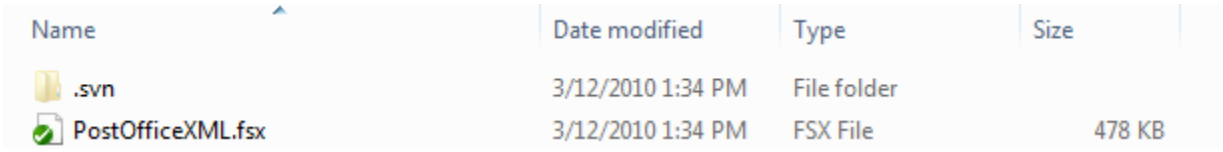


Unless you plan on doing automated changes to the XML file, it's not necessary to know all the details of the file format. In simple terms, the primary tag in FlexSim XML is the <node> tag, representing a node in FlexSim. The node's name is described in the <name> tag, and the node's data, if applicable, is described in the <data> tag. If you do plan on doing automated changes, and need a more detailed definition of the xml format, you can refer to FlexSim's xml schema (see the FlexSim install directory/help/MiscellaneousConcepts/FlexSimXML.xsd for more information).

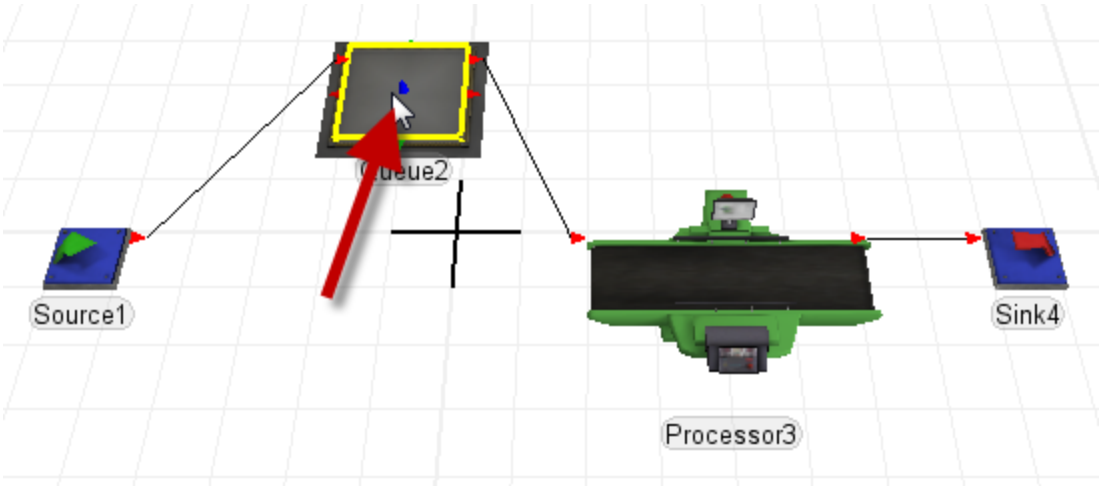
Version Management Utilities

To give you a better idea of the advantages of using an XML format, we'll go through some of the things that you might do with a versioning system. At FlexSim we use Subversion, with TortoiseSVN as our client-side tool, and have found this to meet all of our development needs quite sufficiently. Details of installing and configuring a version management system are outside the scope of this tutorial. You can find many install and setup helps online. As part of this tutorial we will simply assume that, once the model has been saved, a Subversion repository is set up for the model.

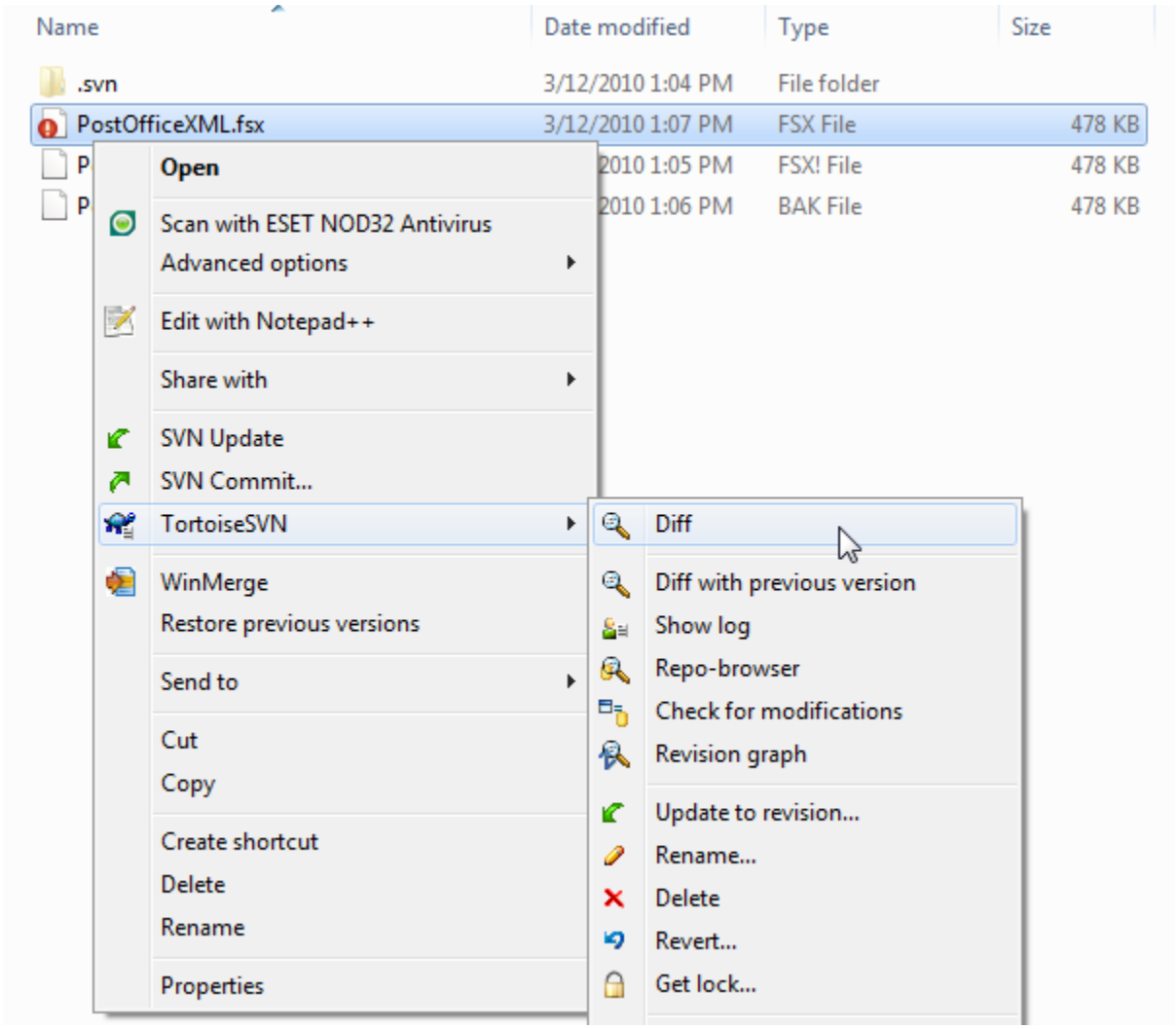
When the Subversion repository is initially created, the file will appear with a green icon indicating that your copy is "clean", or you haven't made any changes since your last check-in.



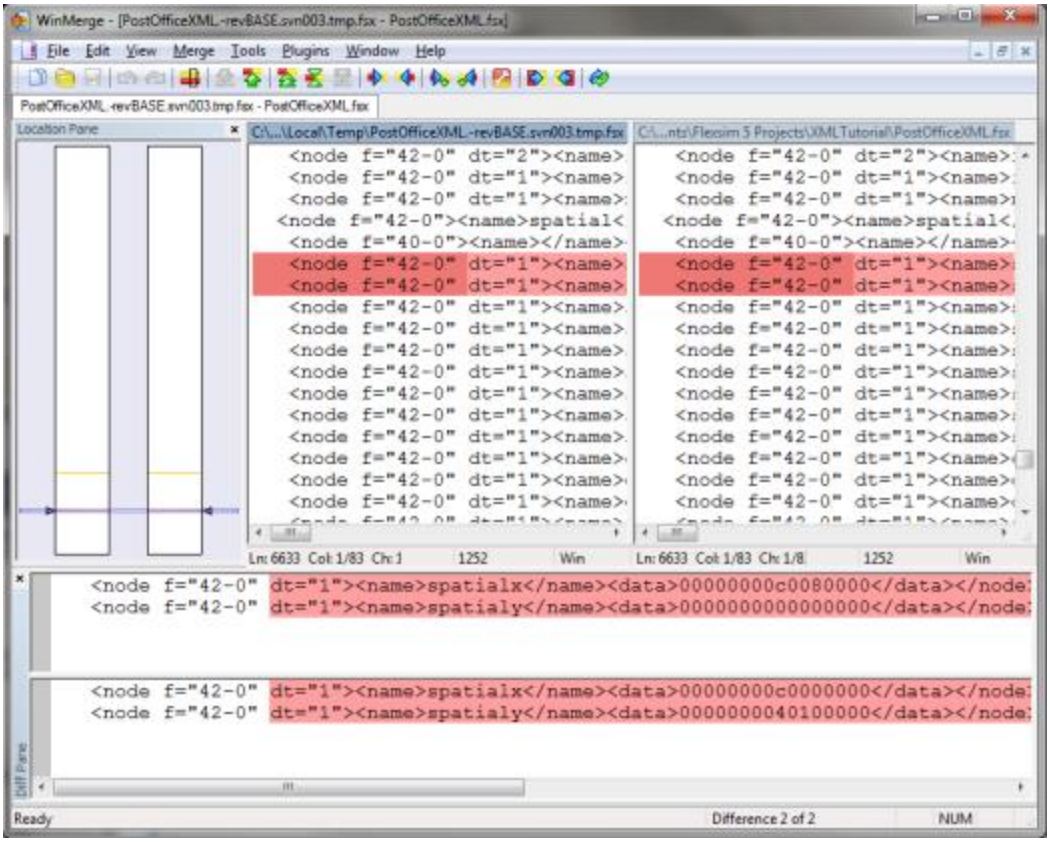
Now let's say we make a small change to the model, such as moving the Queue to a different location and then saving the model again.



Once saved, you'll notice that the file's icon has changed to a red symbol, meaning the file is "dirty", or it has changed since the last check-in. You can also right-click the file, and see exactly what's changed by choosing the "Diff" operation. This will give you a difference comparison of your current copy of the file with the last version that you checked-in.



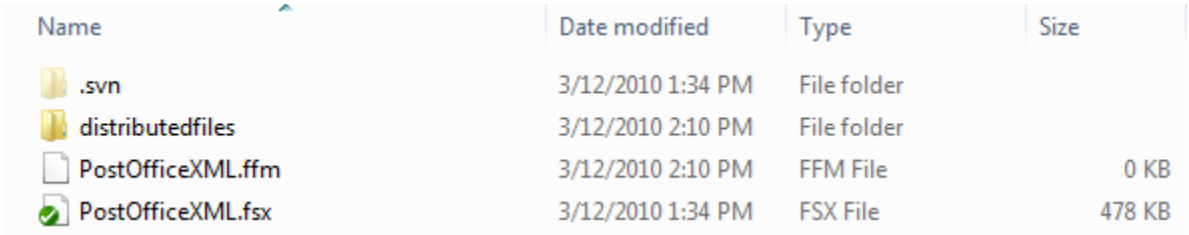
TortoiseSVN provides a simple diff utility, but you can also configure it to use a third-party comparison tool such as WinMerge.



Using the diff tool you can see where the model has been changed, namely in spatialx and spatialy attributes. You can revert line-item changes in the diff tool if you don't want those changes applied.

Distributed Save

Next let's distribute the model across multiple files. To do this you create a "FlexSim File Map" file. This is an xml file with the .ffm extension. It must be placed in the same directory as the model's .fsx file, and, except for the extension, must be given the same name as the .fsx file. So, let's create that file. Let's also create a subdirectory to put the distributed files into.



Now edit PostOfficeXML.ffm in a text editor. The first thing we'll do is put the node model/Tools/active into a different file. This is something you'll probably want to do always if you're using version management. The node model/Tools/active stores all of the windows open in the model, so if you're editing a model and change or reposition the windows that are open, that will change the model file. For version management this is often just static that doesn't need to be tracked, so we'll have the node saved off to a different file, and then we can just never (or at least rarely) check that file into the version management system. Specify the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<flexsim-file-map version="1">
    <map-node path="/Tools/active" file="distributedfiles\windows.fsx" file-map-method="single-node"/>
</flexsim-file-map>
```

Save the file map, then go back into FlexSim. Save the post office model again under the same name "PostOfficeXML.fsx". Now look in the distributedfiles directory. You'll see that it contains a new xml file named windows.fsx. This xml holds the definition of the node model/Tools/active. All interaction with the model remains the same, i.e. from FlexSim you just load and save the main xml model file, but now the model is distributed across multiple files.

In the file map, the main document element is the <flexsim-file-map> tag. Inside the main document element should be any number of <map-node> elements. Each <map-node> element should have a path attribute, a file attribute, and a file-map-method attribute. The path attribute should specify the path, from the main saved node, to the node that is going to be saved into the distributed file. The file attribute specifies the path, relative to the saved model file, to the distributed file that you want to save. The file-map-method should either have a value of "single-node" or "split-points". In this case "single-node" means you want to save all of the active node into windows.fsx.

Now let's do an example of the "split-points" method. Let's say we want to save the Tools folder in one file, the Source and Queue into another file, and the Processor and Sink in a third file. To do this, add another <map-node> element to the file map:

```
<?xml version="1.0" encoding="UTF-8"?>
<flexsim-file-map version="1">
    <map-node path="/Tools/active" file="distributedfiles\windows.fsx" file-map-method="single-node"/>
    <map-node path="" file="distributedfiles\Tools.fsx" file-map-method="split-points">
            <split-point name="Source1" file="distributedfiles\Source_Queue.fsx"/>
            <split-point name="Processor3" file="distributedfiles\Processor_Sink.fsx"/>
    </map-node>
</flexsim-file-map>
```

Now save the file, save your FlexSim model, and again you'll see new files created in the distributedfiles directory.

If a <map-node> element uses the "split-points" method, then it can have <split-point> sub-elements. Each split-point should have a name, defining the name of a node inside the map-node's defined node, and a file attribute defining the file to save to. The map-node has a path="" attribute. This means that it applies to the root node, or the model itself. The map-node's file attribute defines the first file to use. This configuration tells FlexSim to save all sub-nodes in the model up to but not including the node named "Source1" to the file "distributedfiles\Tools.fsx", save everything from Source1 up to but not including the node named "Processor3" in "distributedfiles\Source_Queue.fsx", and save everything from Processor3 to the end in "distributedfiles\Processor_Sink.fsx".

Why Distribute?

The main reason you would want to distribute your model across multiple files is for version management. It allows you to easily see what you've changed in your model by seeing what files, and consequently which parts of the tree, have changed. If you inadvertently changed one piece of the model, you can easily see that and then revert that change if needed. When multiple modellers are developing the model, one modeller can essentially confine himself to one part of the model tree, and by associating that part of the tree with a specific file, it makes it much easier to merge changes from different developers, it reduces the chance of conflicts in the merge, and makes it easier to do a manual merge if needed.

Note on connections: FlexSim's XML save mechanism does have one catch regarding input/output/center port connections, as well as any other mechanism that uses FlexSim's coupling data. If you change connections in one portion of your model, it will actually change the serialized values of all connections/coupling data that occur after those connections in the tree, even if those connections were not changed. This can very easily cause merge issues if multiple modellers are

changing connection data. However, if you distribute the model across multiple files, connection changes where both connection end-points are within the same file will only affect that file, and will not change other files. So if you can localize large "connection sets" into individual files, you can minimize the effect of changes and subsequently minimize merge conflicts.

GUI Events and View Attributes

GUI Events

OnClick	OnKillFocus	OnPreDraw	**The following are not recommended, but may still work**	
OnClose	OnMenuPopup	OnPreOpen		
OnDrag	OnMouseButtonDown	OnPress		OnPreListen
OnDraw	OnMouseButtonUp	OnSelect	OnActivateNotify	OnTimerEvent
OnDrop	OnMouseWheel	OnSize	OnDropFile	OnUndo
OnKeyDown	OnMouseWheelDelta	OnStick	OnDropNode	
OnKeyUp	OnOpen		OnListen	
			OnMessage	

View Attributes

alignrightposition	graphlegend	labelscale	title	viewlighty
alignrightmargin	graphlegendhisto	menucustom	tooltip	viewlightz
alignbottomposition	graphlines	menumain	viewautoconnect	viewmagnification
alignbottommargin	graphmaxpoints	menupopup	viewbackgroundcolor	viewnear
aligncenterx	graphpie	menuview	viewconnectioncolor	viewpointradius
aligncentery	graphpoints	noformat	viewfar	viewpointrx
apply	graphpiedata	objectfocus	viewfield	viewpointry
beveltype	graphstep	pagelist	viewfirstperson	viewpointrz
bitmap	graphtitle	palettewindow	viewfocus	viewpointx
cellheight	graphxy	pickcopydataonly	viewfog	viewpointy
cellwidth	grayed	pickitem	viewfont	viewpointz
close	gridfog	picklist	viewfull	viewprojectiontype
coldlink	gridlinecolor	picklistnameonly	viewhideallbases	viewshowgrid
coldlinkname	gridlinewidth	pickprimary	viewhideallconnectors	viewshowheads
coldlinknamex	gridx	picture	viewhidealldrawcontent	viewsnaptogrid
coldlinkx	gridy	rangeexp	viewhidealllabels	viewsyncupdate
connectorsize	gridz	rangemin	viewhiderouting	viewwindowclean
connectorstyle	hidden	rangemax	viewlightaspos	viewwindowopen
graphannotate	hotlink	spatialsx	viewlightb	viewwindowsource
graphaxes	hotlinkname	spatialsy	viewlightg	viewwindowtype
graphbars	initialtext	spatialx	viewlightr	windowtitle
graphgrid	itemcurrent	spatialy	viewlights	wordwrap
graphhistodata	items	statusbar	viewlightx	

GUI Events

OnClick

This event only applies to FlexSim registered controls. It is fired when the user clicks anywhere inside of the control. This includes when the user clicks the mouse and when the user releases the mouse. There are two access variables for this event:

- "c" is a reference to the control node.
- "i" is a click code: 2 means left mouse button down, 3 means left up, 4 means right down, 5 means right up, and 1 means double-click.

Some commands that you might use within the OnClick are: cursorinfo() or selectedobject().

OnClose

This event contains text data with flexscript code that will be executed when the window is closed.

OnDrag

This event allows you to execute code when an object is dragged from the icongrid onto another view. The event should have text data with flexscript code. Within the function, c gets access to the icongrid, i gets access to the view on which it was dropped, dropx(), dropy(), and dropz() get the drop location if the view is a ortho, perpective, or planar view. dropnodefrom() gets access to the object that was dragged, and dropnodeto() gets access to the object it was dropped onto if one exists. Please note that if no OnDrag event exists, then a copy of the object will be made in the dropped view's focus or in the dropnodeto() if it exists.

OnDraw

This event is fired when the window is drawn. This occurs frequently during a simulation, such as when the window is resized, the model is running, or you click on the window.

OnDrop

This event only applies to FlexSim registered controls. It is fired when the user drags an object from an icon grid and drops it on the view. The attribute should have text data containing flexscript code that fires when the object is dropped. Within the function, you have access to the object that was dragged with dropnodefrom(), and the object that it was dropped onto with dropnodeto().

OnKeyDown

This event only applies to FlexSim registered controls. It is fired when the control has keyboard focus and the user presses a key on the keyboard. There is one access variable, namely c, which is a reference to the control node. You can also query what key went down using lastkeydown(), or query whether any key is down with iskeydown().

OnKeyUp

This event only applies to FlexSim registered controls. It is fired when the control has keyboard focus and the user presses a key on the keyboard. There is one access variable, namely c, which is a reference to the control node. You can also query what key went down using lastkeydown(), or query whether any key is down with iskeydown().

OnKillFocus

This event is fired when the window that the event is applied to loses keyboard focus meaning that you from being able to type in the window to not being able to type in the window because you have clicked on a different window.

OnMenuPopup

This event is executed after the menu is created but before it draws the menu so that you can check or gray out menu items using the commands menucheck() or menugray().

OnMouseButtonDown

This event only applies to FlexSim registered controls. It is fired when the user presses the left mouse button in the view. There is one access variable, namely c, which is a reference to the control node.

OnMouseButtonUp

This event only applies to FlexSim registered controls. It is fired when the user releases the left mouse button in the view. There is one access variable, namely c, which is a reference to the control node.

OnMouseWheel

This event only applies to FlexSim registered controls. It is fired when the control has keyboard focus and the user scrolls the mouse wheel. The control also needs to have an OnMouseWheelDelta event with number data. When the user scrolls the mouse wheel, FlexSim will set the value of the OnMouseWheelDelta value according to how much the user has scrolled, and then will call the OnMouseWheel function. Within the function, c accesses the OnMouseWheel event itself.

OnMouseWheelDelta

This event is used as described in the OnMouseWheel event above.

OnOpen

This event allows you to specify flexscript functionality that will fire when the window is opened. It is fired when the window is initially opened, as well as when the window is restored after a compile, as well as when the window is "redirected" to point to a new object if the user switches the window to point to another object. You may use this trigger to initialize settings in controls that may not have a coldlink.

OnPreDraw

Executed before an OnDraw event. See OnDraw for more information.

OnPreOpen

This event allows you to execute functionality before the window is created. The OnPreOpen is fired after the tree structure for the window is created, but before the window itself is initialized. Unlike OnOpen, it is not executed after a compile or when the window is redirected. You could use the OnPreOpen to modify the structure of the window before it is opened, such as adding or removing tab windows, or adding or removing any controls from the tree structure of the window.

OnPress

The OnPress event specifies code that will be executed when the button is pressed. It should have text data containing flexscript code. Within the code, c accesses the button view.

OnSelect

The OnSelect event is a trigger that fires when the user selects an option in the combobox, the user drags the locator of a tracker to a given position, or when a tab page is selected for a tabcontrol. The event should have text data with flexscript code specifying what to do when the user selects the option. Within this trigger you will want to access the value of the itemcurrent attribute to find out which option the user selected. For the Tracker, within the function, c will access the tracker control while for the Tabcontrol c accesses the tabcontrol view. For a tabcontrol, use get(itemcurrent(c)) to get the currently selected page.

OnSize

This event is executed when the window gets re-sized.

OnStick

This event is executed on joystick or 3D mouse events (moving the joystick or pressing buttons on the joystick or 3D mouse).

OnActivateNotify

Executed when you change the active window. This event may only work on palette windows. An example of this event can be found in the Tree Navigation window. When a 3D view is selected with the Tree Navigation window open, options in the Tree Navigation window are grayed out. However, when a tree window is selected, those options become active (un-grayed) again.

OnDropfile

Executed when a file is dragged on top of the window and then dropped onto it.

OnDropNode

This event should contain text data and defines flexscript code that will be executed when the user holds a key down while click-dragging from one object drawn in the view to another.

OnListen

Executed on any event happening: after dispatch, when event has been removed. To set up listening, you add a "listeners" attribute to the object you want to listen to (Object A). This listeners attribute should have subnodes that are couplings to the object(s) that you want to listen from (Object B). Before an event fires on Object A, Object B's OnPreListen event function will fire. After an event first on Object A, Object B's OnListen event function will fire. You can also filter events by adding a subnode with number data below Object B's coupling node. The number data should be a bitwise sum of the bitshifted event codes you want to listen for. If the coupling has no subnode, the object will default to listening to every event. To help you with the OnPreListen and Onlisten event functions, you may consider using the listenerinfo(int info) command. The information returned is information that was passed to the event you are listening to. Info 1 will return the engine event code. Engine event codes have macros such as SM_MESSAGE and SM_DRAW. Info 2 and 3 return pointers to associated

OnMessage

Executed when the GUI object receives a message that was sent from another object in the model.

OnPreListen

Executed on any event happening: before dispatch, when event is still in list. Refer to OnListen for a more detailed description.

OnTimerEvent

Executed on timer event. Executed on an event that was created with createevent(). This occurs in numerous objects, one example is the Processor finishing its process.

OnUndo

Executed on content-defined undo action. When you call the createundorecord(.,.,UNDO_CUSTOM); command you create a document that records the most recent changes that were made to the model. Then when you undo something (Ctrl z) you can use the OnUndo event to reference that information and undo what was done or perform some other function.

View Attributes

alignbottommargin

These attributes signal that the control's margin is "locked" to the right or bottom margin of the control's container window. They should contain number data, and the value represents the offset distance from the container window's right or bottom margin, in pixels, that the control's margin will be "locked" to. For example, if you give a button control an alignrightmargin attribute with a value of 10, then as you resize the window, the button will automatically resize so that its right margin is 10 pixels from the right edge of the window.

aligncenterx

These attributes signal that the control should be center aligned with its container control. The attributes should contain number data, and their value represents an offset from the center position.

aligncentery

These attributes signal that the control should be center aligned with its container control. The attributes should contain number data, and their value represents an offset from the center position.

apply

If the apply attribute is added to a button, then FlexSim will call applylinks() on the button's owner view when the button is pressed. The attribute needs no data.

The beveltype attribute specifies what the border of a panel should look like. The attribute should have number data with a value between 0 and 2. A value of 0 will cause no border to be drawn. A value of 1 causes a one pixel sunken border to be drawn. A value of 2 causes a 2 pixel border to be drawn.

bitmap

Using a bitmap attribute, a bitmap image can be applied to a given view/control.

Panel: Simply add the bitmap attribute to the panel and give it the path to the bitmap file (like button\up_arrow.bmp), then give the panel a viewfocus attribute with the following text: "..>bitmap". This will cause the bitmap to be shown on the panel.

Static: The bitmap attribute causes the static control to show a bitmap instead of text. The attribute should have text data that defines a path to the bitmap file, starting at the FlexSim main directory. The file must be a .bmp file. You can also specify within the bitmap file certain areas a "transparent",

meaning the standard background color of the view will show through. To do this, the bitmap must be created in index color mode 24 bits per pixel, and the color that Windows will designate as transparent is the color R: 192 G: 192 B: 192.

Checkbox: The bitmap attribute causes the checkbox to have a bitmap next to it instead of text. It should be created the same as the static view type was created.

Radiobutton: The bitmap attribute causes the radiobutton to have a bitmap next to it instead of text. This one is also created in the same way as the static view type was created.

Button: The bitmap specifies a bitmap to be shown on the button. It should contain text data specifying a file path to the bitmap, starting at the FlexSim main directory. If the bitmap attribute is present then the name of the button will not be shown.

cellheight

This attribute establishes the height of a given cell. It can be used in a couple of different views.

Table: This attribute specifies the height, in pixels, of each row in a table. Unlike table columns however, all table rows must be the same.

Icongrid: This attribute allows you to define the height, in pixels, of each rectangle in the icongrid. It should have number data containing the desired height.

cellwidth

This attribute establishes the width of a given cell. It can be used in a couple of different views.

Table: This attribute specifies the default column width, in pixels. The attribute should have number data with the value in pixels. The attribute can also contain subnodes. Each subnode should have number data that defines the column width of an individual column. The first subnode defines the column width of the row header column while the second subnode defines the first column's width, and so forth.

Icongrid: The attribute allows you to define the width, in pixels, of each rectangle in the icongrid. It should have number data containing the desired width.

Tree: The attribute lets you specify how wide, in pixels, each node's name will be shown.

close

For a Button, if the close attribute is added, then FlexSim will close the button's owner view when the button is pressed.

coldlink

This attribute is used to link a control with a value in the model. For example, an edit control with a coldlink to the max content (or another value of an attribute or variable node) of a queue will show the max content as a text in the edit control. It is a "cold" link because it is only refreshed when the window is opened, and only applied when the Apply or OK button is pressed. The coldlink attribute should contain text data that is a path to the node that holds the linked data. The path starts at the coldlink node itself. You can use the applylinks() command to apply or refresh the coldlink. The first parameter is a node that is the start location for a recursive search. The second parameter is optional. The command will recursively search the window's tree structure and find any coldlinks (and hotlinks) and apply the coldlinks to the object's attributes. If the optional second paramter is 1, then instead of applying the coldlinks to the object, the applylinks() command will refresh the window's values as defined by the object's values. Below is a list of the different areas of GUI building that use coldlinks and the attributes purpose in each case. For more information on traversing the tree, see Traversing the Tree.

Edit View: This attribute defines what will appear in the text of the control. If the linked code contains number data, then the value will be copied into the control's text with the precision that the user specified in the main Edit menu. If you were to use a coldlink for a static you would find the attribute

allows you to have the text be dynamic based and an attribute of the object.

Static View: This attribute simply defines the text of the static control. Because the text of the static control is not editable, the links are not applied as they would be for other controls. You can also use the setviewtext() and getviewtext() to explicitly get and set the text of the control. This can similarly be done for an Edit control as well.

Checkbox: This attribute links the state of the checkbox to a value in an attribute of the object. The node that it links to should hold number data and have a value of 1 or 0. 1 will cause the box to be checked, 0 will cause the box to be unchecked. You can also explicitly check the box with setchecked() or get whether it is checked with getchecked() (1 means checked, 0 means unchecked).

Radiobutton: The attribute links the state of the radibutton to a value in an attribute of the object. The node that you link to should hold number data and have a value of 1 or 0. 1 will cause the button to be checked, 0 will cause the button to be unchecked). Radio buttons present a problem because in order to have them work using just a coldlink, you would need to connect each one to an individual node's value. Hence you would need 5 nodes in the object's attributes with only one having the value of 1 and the rest having 0. What users often want instead of this is to have a set of radiobuttons that reflect one value. For example, if a node has value 1, then radio button 1 should be checked; for value 2, radio button 2 should be checked, and so forth. In order to do this you would need to write your own code in a coldlinkx or in the OnOpen of the window. Alternatively, you can use a combobox control. The combobox control fulfills the same functional requirement of choosing one of several options, but packages it all into one control that can be linked to a value on one node.

Button: The coldlink works as stated at the beginning and simply designates what will be shown as the button's text.

Script: The attribute simply links the text of the script control to the text of a node on the object and works like documented earlier.

Tracker: The attribute simply links the tracker or statusbar to a value on a node and it works like documented earlier.

Statusbar: The attribute simply links the tracker or statusbar to a value on a node and it works like documented earlier.

coldlinkname

This attribute is just like the coldlink, except it links with the name of the node specified by the coldlink's path. For more information on traversing the tree, see Traversing the Tree.

coldlinknamex

This attribute is just like the coldlinkname, except that instead of holding text data with a path to the involved node, the coldlinknamex holds flexscript code that returns the node whose name should be linked to. For more information on traversing the tree, see Traversing the Tree.

coldlinkx

This attribute is like a coldlink, except that instead of holding text data with a path to the involved node, the coldlinkx holds flexscript code. The flexscript function should return a reference to the node that the view should link to. If 0 is returned, then nothing will be applied or refreshed for the control. Within the function there are 3 access variables. c is a reference to the control itself. i is a reference to the object focus of the view (the same as node("@>objectfocus+",c)). Eventdata is either 1 or 0. If 0, then the coldlinks function is being executed in order to refresh the control according to the object's variable. If 1, then the coldlinkx is being executed in order to apply the value to the object's variable. Please note that when the coldlinkx function is called, the return value, or in other words the reference to the linked node, is not remembered by the window. Each time the control needs to be refreshed or applied, the coldlinkx function is called

again. This means that the coldlinkx function can actually be called many times throughout the life of the window. For more information on traversing the tree, see Traversing the Tree.

connectorsize

This attribute determines the size of the arrows drawn at the ends of connections between connected objects in a 3D view.

graphaxes

In a Graph, if this attribute is present, then the grid will show the min and max values of the x and y range for the graph. The attribute does not need any data.

graphbars

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphbars and the attributes purpose in each case.

Histogram: This attribute signifies that the graph will be shown as bars. The attribute needs no data.

Line Chart: Add this attribute to have a bar drawn from 0 to the y value at each point in the graph. The attribute needs no data.

Bar Chart: Add this attribute to have a bar drawn from 0 to the y value at each point in the graph. The attribute needs no data.

Scatter Plot: Add this attribute to have a bar drawn from 0 to the y value at each point in the graph. The attribute needs no data.

graphgrid

In a Graph, if this attribute is present, then a grid will be shown as a background for the graph. The width of the grid does not correlate to any set value range in the graph, but is rather for comparative purposes, for example to compare the height of two bars in the graph. The attribute does not need any data.

graphhistodata

In a Histogram, If this attribute is present, then the graph will show a histogram. The attribute should have text data containing a path to a node that contains "bucket" sub-nodes. The number of sub-nodes should be the number specified by the viewfocus plus 2. The first sub-node will be designated as the "underflow" node, where any values less than the minimum range value of the histogram will be added to this node. The last node is "overflow" for values that are greater than the histogram's maximum range. Every other sub-node represents the histogram value for the "bucket" for the sub-node's corresponding interval.

graphlines

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphlines and the attributes purpose in each case.

Line Chart: Add this attribute to have a line drawn between consecutive points in the graph. The attribute needs no data.

Bar Chart: Add this attribute to have a line drawn between consecutive points in the graph. The attribute needs no data.

Scatter Plot: Add this attribute to have a line drawn between consecutive points in the graph. The attribute needs no data.

graphpie

In a Pie Chart, this attribute signifies that the graph will be shown as a pie chart. The attribute needs no data.

graphpoints

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphpoints and the attributes purpose in each case.

Line Chart: Add this attribute to have a dot drawn at each point in the graph. The attribute needs no data.

Bar Chart: Add this attribute to have a dot drawn at each point in the graph. The attribute needs no data.

Scatter Plot: Add this attribute to have a dot drawn at each point in the graph. The attribute needs no data.

graphpiedata

In a Pie Chart, this attribute should have text data the contains a path to a node with sub-nodes. Each sub-node's name will be show in the pie chart's legend, and should have number data that is the value for that item in the pie chart. The sum of all sub-nodes of the focus node will represent a full 360 degree circle, and the each sub-node's value represents a chunk of the pie.

graphstep

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphstep and the attributes purpose in each case.

Line Chart: If this attribute and the graphlines attribute are present, then each line will be drawn first horizontally and then vertically to the next point, creating a stair-step effect instead of diagonal lines.

Bar Chart: If this attribute and the graphlines attribute are present, then each line will be drawn first horizontally and then vertically to the next point, creating a stair-step effect instead of diagonal lines.

Scatter Plot: If this attribute and the graphlines attribute are present, then each line will be drawn first horizontally and then vertically to the next point, creating a stair-step effect instead of diagonal lines.

graphtitle

In a Graph, if this attribute is present, then the grid will show a title for the graph. This is usually the text data on the viewfocus node except in the case of a histogram, where it is the text data on the graphhistodata focus. The attribute does not need any data.

graphxy

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use graphxy and the attributes purpose in each case.

Line Chart: Add this attribute to make each sub-node pair be interpreted as an x/y value. If this attribute is not present, then each sub-node (not sub-node pair) will be interpreted as a y value, and the points will be evenly distributed across the x-axis, the first point with the x value 1, the second with x-value 2, and so forth. You may want to leave this attribute out if you want to display a bar chart where the bars are evenly distributed across the x axis.

Bar Chart: Add this attribute to make each sub-node pair be interpreted as an x/y value. If this attribute is not present, then each sub-node (not sub-node pair) will be interpreted as a y value, and the points will be evenly distributed across the x-axis, the first point with the x value 1, the second with x-value 2, and so forth. You may want to leave this attribute out if you want to display a bar chart where the bars are evenly distributed across the x axis.

Scatter Plot: Add this attribute to make each sub-node pair be interpreted as an x/y value. If this attribute is not present, then each sub-node (not sub-node pair) will be interpreted as a y value, and the points will be evenly distributed across the x-axis, the first point with the x value 1, the second with x-

value 2, and so forth. You may want to leave this attribute out if you want to display a bar chart where the bars are evenly distributed across the x axis.

grayed

This attribute causes the control to be disabled, graying the control and disallowing the user from manipulating the control. It should container number data; 1 means grayed, 0 means not grayed. Once a control has been initialized, the "grayed" state of the control will not change by simply changing the value of the grayed attribute. Because of this, and because the "grayed" state of a control is very dependent on certain parameters of objects, and can change during the life of the window, it is usually more practical to use the windowgray() command to change the "grayed" state of a control. This command does not require you to even have a grayed attribute, so the grayed attribute, for the most part, is unneeded.

gridfog

This attribute determines how much fog will cover the grid on a scale from 0 to 1. Typically small numbers work better than larger ones.

gridlinecolor

This attribute determines the color of the gridlines. The 3 subnodes of this attribute determine the red, green, and blue color components used to draw the grid in a 3D view.

gridlinewidth

This attribute determines the width of the grid lines drawn in a 3D view.

gridx

If the graphgrid attribute is present, then you can also add a gridx attribute, which specifies the x grid interval.

gridy

If the graphgrid attribute is present, then you can also add a gridy attribute, which specifies the y grid interval.

gridz

If the graphgrid attribute is present, then you can also add a gridz attribute, which specifies the z grid interval.

hidden

This attribute designates that the control will be hidden from the user. For the same reasons as with the grayed attribute, this attribute is usually unneeded and can be replaced with the windowshow() command.

hotlink

This attribute is used to link a control with a value in the model, just like the coldlink mentioned above. It is a "hot" link because the value shown in the control is continuously refreshed each time the window is repainted. Otherwise the hotlink is exactly the same as the coldlink. For more information on traversing the tree, see Traversing the Tree.

hotlinkname

This attribute is just like the hotlink, except it links with the name of the node specified by the coldlink's path. It can be used in a static or edit control in the same way you would use a coldlinkname. For more information on traversing the tree, see Traversing the Tree.

hotlinkx

This attribute can be used in a static, edit, or checkbox control in the same way you would use a coldlinkx, except that it will dynamically update like a hotlink. For more information on traversing the tree, see Traversing the Tree.

itemcurrent

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use itemcurrent and the attributes purpose in each case.

Check Box: The itemcurrent attribute is actually added to the checkbox when an instance is created. Usually it signifies whether the box is checked, but it may not always. Use getchecked() to get the check state of the box.

Radio Button: Again, the itemcurrent attribute is added to the radiobutton automatically when an instance is created. Usually it signifies whether the box is checked, but it may not always. Use getchecked() to get the check state of the box.

Combobox: The itemcurrent attribute is also a required attribute. It should have number data, and it specifies which attribute is currently selected.

Listbox: The itemcurrent attribute is also required. It also acts just like the itemcurrent attribute of the combobox.

Tracker: The itemcurrent attribute is a required attribute. Its value holds the current value that has been selected for the tracker.

Tabcontrol: The itemcurrent attribute is required. It allows you to know which tab is currently selected. It should contain number data whose value will be set whenever the user chooses a tab.

items

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use items and the attributes purpose in each case.

Combobox: The items attribute is a required attribute. It should contain subnodes. Each subnode represents an option in the combobox. The subnode's name specifies the text that will be shown in the option. You can manually add subnodes and give them names if your combobox options do not change. Often you will want the options in the combobox to be dynamic. You can do this by writing code in an OnOpen or some other trigger that populates the items attribute dynamically. Access the items attribute with the items() attribute command, clear the contents of it, then add nodes into the items attribute with nodeinsertinto(), then set their names with setname(). Once you've populated the list, set the itemcurrent attribute's value to the rank of the option that you would like the combobox to reference by default, then call comborefresh() to refresh the combobox windows options according to the new items list.

Listbox: The items attribute is required on the listbox. It represents the list of options in the list box. It acts just like the items attribute in the combobox.

menucustom

In a dialog view, this attribute acts much like the menupopup attribute, except the menu will appear at the top of the window instead of appearing when the user right-clicks in the view.

menupopup

This attribute is only applicable for FlexSim registered controls. By adding this attribute you can define the menu that will appear when the user right-clicks on the control. The attribute should contain sub-nodes. Each sub-node is a menu option of the pop-up. The sub-node's name defines what the menu item's text will be. The sub-node should have text data. The text data defines a flexscript function that will be executed when the option is selected. Within the flexscript function there is one access variable, c. c is a reference to the menu option sub-node. Some commands that might be used within the flexscript function are listed below. For detailed information on each command, refer to the command documentation.

menuview

In a dialog view, this attribute allows you to have the standard FlexSim menu appear at the top of the window. The attribute should have number data and be set to 1 to have the menu appear.

noformat

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use noformat and the attributes purpose in each case.

Script: The noformat attribute causes the script control to be unformatted, meaning no code highlighting, line numbers, or folding will be done to the text of the control.

Table View: Like the script control, the table control uses the Scintilla text editor. By adding the noformat attribute, Scintilla features like code highlighting, line numbers, folding, etc. are disabled. Usually this attribute should be present as it makes quick table editing easier.

Tree View: If this attribute is present, then when text data is edited it will not be formatted for code.

objectfocus

In a dialog view, this attribute will specify a path to the object that a given window instance will point to. The createview() command contains a parameter that specifies the objectfocus, the objectfocus attribute will be filled with that parameter when the window is opened.

pagelist

In a tabcontrol view, the pagelist attribute allows you to have an external source for the pages of the tab control. The attribute should have text data specifying a path to a node that contains subnodes. Each of the subnodes should contain text data that is a path to the page control. The pagelist attribute is used quite often in object Properties windows.

palettewindow

In a dialog view, this attribute will cause the window's title bar to be smaller and look more like a tool window. The window will also always appear on top of other windows. The attribute needs no data.

pickcopydataonly

In a combobox view, the pickcopydataonly attribute causes the combobox to execute special functionality that allows for the implementation of FlexSim code pick lists. Although it could be used for other purposes, the main purpose is for code pick lists. In simple terms, the attribute causes the text of a given items subnode to be copied to a linked variable on the object. If you add the pickcopydataonly attribute, you will need to also add a pickprimary attribute and a picklist attribute. The pickprimary attribute acts much like a coldlink attribute. It links the combobox to an attribute of the object. When the combobox is initialized, the template code for the node referred to by pickprimary is copied into the combobox's items attribute as the first, or default, option. Then the referenced picklist is copied into the items attribute as additional options for the combobox. When the links are applied, then the text of the currently selected item is copied back to the node referenced by pickprimary. Note here that the subnodes of the items attribute now will have text data, and whatever item is picked, its text data is copied to the pickprimary reference. The existence of the pickcopydataonly attribute also affects some additional features with regards to the picklist attribute. The picklist attribute can contain subnodes that refer to other pick lists. If present, the subnodes should contain text data that is a path to a pick list, just like the picklist attribute itself. These pick lists will be appended on to the options of the combobox. Thus you can have the combobox provide a concatenated list of options from several different locations. The need for this feature came up because often different pick options may share a general list of options, but then have a unique list as well. For example, a setup time pick list may have all of the same options as the cycle time pick list, but it also needs a "Setup time if itemtype changes" option. So this feature allows the setup time pick list to mainly use the cycle time's "shared" pick list, but then add an additional option specific to a setup time. Another feature that is available when the pickcopydataonly attribute is used is with using code headers. If the main pick list node that is referred to by the picklist attribute contains text data, then that text data will be appended to the front of any of the options. This allows for a common header section for all options, and lets the options themselves just

implement the relevant code and not worry about the headers. This also allows for easy maintenance if the header changes. For example, a header section may contain the code: "treenode current = ownerobject(c);". This code will be appended in front of the actual code for a given option. This method is used for all standard pick list comboboxes in FlexSim, so you can find examples of this quite easily. You can also give the combobox a picklistheader attribute with text data designating the header code to use instead of what is found on the referenced pick list node.

pickitem

In the combobox view, the pickitem attribute causes the combobox to be linked to a number value on the object. The attribute does not need any data. If you give the combobox this attribute, then you will also need to give the combobox a coldlink attribute. For example, if you give the combobox a pickitem attribute and a coldlink with the text: @>objectfocus+>variables/arrivalmode, then this will link the combobox with the arrivalmode variable on the object. If the user chooses the second option in the combobox, then the arrivalmode variable will be given a value of 2. A third option selected results in a value of 3, and so forth. As an example, the Source's Properties page uses this attribute to tie the "Arrival Style" combobox to the Source's arrivalmode variable.

picklist

In the combobox view, the picklist attribute allows you to redirect where the list of options is located. You may want to do this if you use several comboboxes in different GUIs, but all of the comboboxes use the same list of options. The picklist attribute should have text data which is a path to the pick list, starting at the picklist attribute. The pick list that is referred to should be structured just like the items attribute, with subnodes whose names designate the option text. When the window is opened, the referenced pick list will be copied into the items attribute of the combobox.

pickprimary

In the combobox view, this attribute is used with pickcopydataonly, as explained above.

picture

In the icongrid view, this is used in conjunction with viewfocus, and should contain a path to a picture file that will be displayed in the icongrid.

rangeexp

In the tracker view, this attribute tells the tracker to exponentially increase the value as the locator is dragged to the right, instead of linearly. It should have number data specifying the factor to exponentially increase by. The run speed control at the bottom of the FlexSim window uses this attribute.

rangemin

In the tracker view, the rangemin attributes specify the minimum values of the tracker. They should have number data specifying the minimum value.

rangemax

In the tracker view, the rangemax attributes specify the maximum values of the tracker. They should have number data specifying the maximum value.

spatialsx

This specifies the X size of the control it is on.

spatialsy

This specifies the Y size of the control it is on.

spatialx

This specifies the X location of the control it is on.

spatialy

This specifies the Y location of the control it is on.

statusbar

If this attribute is added to a 3D view and has a numerical value of 1, then a status bar will be drawn at the base of the view. An example of this can be found at the base of the standard 3D view.

tooltip

This attribute defines a tip that will pop-up when the mouse hovers over the control for a certain amount of time. The attribute should have text data containing the text that should be shown.

viewbackgroundcolor

This attribute defines the text background of the control. For a 3D or planar view, it defines the background color of the scene. The attribute should have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively. This attribute can be added to almost every view.

This attribute specifies the connection colors. The attribute should have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively.

viewfar

This attribute defines the far clipping plane distance.

viewfield

This attribute defines the field-of-view angle in degrees.

viewfirstperson

This attribute uses the firstperson camera mode in a view.

viewfocus

This attribute has many diverse uses and purposes in GUI building. Below is a list of the different areas of GUI building that use viewfocus and the attributes purpose in each case.

Graph View: The viewfocus attribute is required. It should have text data containing a path to a node that contains data for the graph. The structure of the data depends on the type of graph. If the graph is a pie chart, then the view focus node should have text data that specifies the name of the chart. If the graph is a line graph, scatter plot, or bar graph, then again the viewfocus node should have text data that specifies the title of the chart, and the viewfocus should have sub-nodes. Every odd-numbered sub-node will represent a point on the x axis, and every even-numbered sub-node represent its corresponding y value. If the graph is a histogram, then the viewfocus node should have three sub-nodes, each having number data. The first sub-node should be the minimum value of the histogram range. The second sub-node should be the maximum value of the histogram range. The third sub-node should be the number of "buckets" or divisions in the histogram.

Icongrid View: The viewfocus attribute is a required attribute. It should have text data that contains a path to the view focus of the icon grid. The icongrid will display each subnode of the view focus as a drag-able rectangle. In order for a given sub-node to be displayed, it must have object data, and it must have a picture attribute. If not, the object will not be shown at all in the icongrid. The picture attribute of the object may contain text data that specifies a path to a bitmap file, starting in the FlexSim main directory. If the path is valid, then the picture will be shown in the object's rectangle. Otherwise, only the name of the object will be shown.

Ortho/Perspective View: This attribute defines what the ortho view is "looking at". It should contain text data with a path to the node containing the objects to be viewed.

Planer View: This attribute defines what the planar view is "looking at". It should contain text data with a

path to the node containing the objects to be viewed.

Table View: The viewfocus attribute is required for the table control. It specifies what the table will "point at". It is similar to the objectfocus attribute. It should have text data that is a path to the node that contains the table information. There are two options for the structure of the table node itself. It can be a list, meaning it contains a number of subnodes. If the table node is a list, then the table control will show just one column of data, and the text in each entry is the data, either text or number, of each of the subnodes in the list. The row headers of the table are defined by the names of each subnode. The other option is to have the table node be an actual table. In this case the table node contains a set of subnodes. Each subnode is one row in the table, and it contains subnodes, each being an individual entry in the table. The number of columns that will be shown in the table is defined by the number of subnodes of the first row. Row headers again are defined by the names of each row node. Column headers are defined by the names of the subnodes of the first row node. Please note that with a table, it is a direct view into the table data. This means that any changes made in an entry in the table will be applied as soon as you click off of the entry. This is different than using coldlink in other controls, where changes are only made when the user hits the Apply button. This is why the table control uses the viewfocus attribute, because it is a direct view into the data of a model, just like an ortho or perspective control.

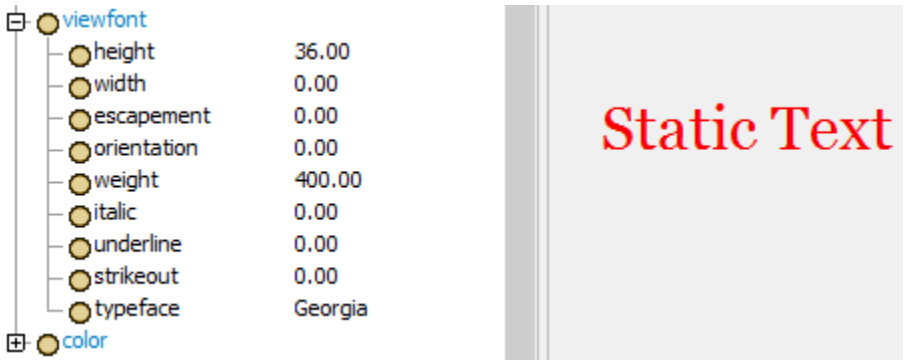
Tree View: The viewfocus specifies the focus of the tree view. It should have text data with a path to the node that will be the head of the tree.

viewfog

This attribute defines the fog density factor. It is normally a number between 0 and 1, where 0 is equivalent to none and 1 is equivalent to the maximum value.

viewfont

This attribute only applies to Windows Common Controls. It defines the font that the control will use. The viewfont attribute is a wrapper around the Windows API CreateFont() command. Add this attribute and give it sub-nodes defining the different parameters that will be passed into CreateFont(), as shown in the figure below. You can refer to Microsoft's Windows API documentation for more information on this.



viewfull

This attribute is used to specify that the window it is placed is does not have a title bar.

viewhideallbases

This attribute hides all objects 2D shapes in a view.

viewhideallconnectors

This attribute hides all port connectors in a view.

viewhidealldrawcontent

This attribute hides all 3D shapes in a view.

viewhidealllabels

This attribute causes the objects' names not to be shown, but only their pictures. The attribute should have number data with a value of 1.

viewhiderouting

This attribute hides all object routing lines in a view.

viewlightaspos

This attribute sets the light as directional or a point. To set it the light to directional, the value needs to be 0. To set it the light to a point, the value needs to be 1.

This attribute must be the parent of a list of nodes each having viewlightx, viewlighty, viewlightz, viewlightr, viewlightg, viewlightb, viewlightaspos attributes. The subnodes define the position and color of directional lights or point light sources used in the 3D view.

viewlightr

This attribute corresponds to the red color component of the light.

viewlightg

This attribute corresponds to the green color component of the light.

viewlightb

This attribute corresponds to the blue color component of the light.

viewlightx

This attribute sets the x position of a point light source or the x component of a directional light source.

viewlighty

This attribute sets the y position of a point light source or the y component of a directional light source.

viewlightz

This attribute sets the z position of a point light source or the z component of a directional light source.

viewmagnification

This attribute is only used with ortho views and planer views, and specifies the magnification or "zoom" of the view.

viewnear

This attribute defines the near clipping plane distance.

viewpointradius

This attribute is only used with perspective views, and specifies the camera's distance from the focal point.

viewpointrx

This attribute specifies the RX viewing angle of the view.

viewpointry

This attribute specifies the RY viewing angle of the view.

viewpointrz

This attribute specifies the RZ viewing angle of the view.

viewpointx

This attribute specifies the X focal point of the view.

viewpointy

This attribute specifies the Y focal point of the view.

viewpointz

This attribute specifies the Z focal point of the view.

viewprojectiontype

The viewprojectiontype attribute specifies whether the view is an ortho or a perspective view. It has number data, 1 signifying ortho, 0 signifying perspective.

viewshowgrid

This attribute enables or disable the showing of grid in a view. It should have number data with the value of 1, meaning show the grid and the value of 0, meaning don't show the grid.

viewshowheads

This attribute shows or hides all node ranks and node indices in a view. It should have number data with the value of 1, meaning show the node ranks and the value of 0, meaning hide the node ranks.

viewsnaptogrid

This attribute enables or disable the grid snapping in a view. It should have number data with the value of 1, meaning enable snap to grid and the value of 0, meaning disable snap to grid.

viewsyncupdate

This attribute enables or disable the repainting of all views when object editing is done. It should have number data with the value of 1, meaning enable repainting and the value of 0, meaning disable repainting.

viewwindowclean

This attribute specifies that when the window is closed, the underlying view tree is to be deleted. It is automatically managed by the FlexSim engine.

viewwindowsource

This attribute is required if you want the icongrid to be draggable. It should have number data with the value of 1, meaning yes you want the user to be able to drag objects from the icongrid onto other views.

viewwindowtype

This is the type of the window. Window types have specific values associated with them. When you create a window or drag out a control on the window it is given this attribute with a numeric value representing the window type. Window types are associated with window controls like buttons, radio buttons, check boxes, edits, listboxes, etc. As well as controls like groupbox, panel, table, graph, tree icongrid, etc. All the window types can be found in the online documentation in the View Attributes Reference section.

windowtitle

This attribute defines the title of the window. It should have text data defining the title. Otherwise the viewfocus path is used.

wordwrap

This attribute specify wrapping text in the editor. If the value is 0, then there is no text wrapping.

View Attributes Reference

This document describes in detail the different GUI attributes you can use when creating custom GUIs. This describes attributes used for each view type and how those attributes affect the behavior of the control. Please read the topic on Graphical User Interfaces before referencing this topic.

Topcis

- **View Window Classes**
- **Windows Common Controls**
- **FlexSim Registered Controls**
- **Other Controls Used by FlexSim**
- **General Attributes**
- **Control Specific Attributes**

View Window Classes

In this topic, we will occasionally refer to FlexSim window classes versus Windows common controls. The difference between these two can be subtle but are nonetheless important. FlexSim uses the standard Microsoft Windows interface for its windowing system. Within that interface there is an ability to register custom window classes and define special functionality for those windows. There is also the ability to use Windows common controls, which are standard window classes that have already been defined by Windows like buttons, comboboxes, etc. FlexSim uses many of the common controls, but it also has created some of its own window classes. These two categories, and which FlexSim view types fit in which category, are listed below. There are several view attributes in the "general" category that will only work on FlexSim registered controls. This will be mentioned under each attribute's description.

Tthe terms view and control are used interchangeably. They basically describe a window control in FlexSim.

Many of the attributes described below can contain flexscript code that is executed. Unless otherwise stated, when adding such a flexscript attribute, you can choose whether you want to toggle the node as flexscript. If you toggle the node as flexscript, then FlexSim does not need to build the node's flexscript code when the function is executed. This results in a faster execution time, but also requires more memory to store data in the tree, and it causes FlexSim's global "Build all flexscript" function to take more time. As a rule of thumb, if the function is executed in "user" time, meaning it is an operation that is explicitly done by the user, like pressing a button or selecting a combobox option, FlexSim can build the flexscript fast enough so the user notices no difference, so you don't need to toggle the node flexscript. However, if it is an operation that is executed quite often, like a hotlinkx, or an OnDraw, then it might be wise to toggle the node as flexscript to increase refresh rates.

Windows Common Controls

- **Button**
- **Checkbox**
- **Combobox**
- **DateTimePicker**
- **Edit**
- **Listbox**
- **Radiobutton**
- **Scrollbar**
- **Spincontrol**
- **Static (or label)**
- **Statusbar**
- **Tabcontrol**
- **Trackbar**

- **Treeview**

FlexSim Registered Controls

- **3D View (Ortho/Perspective)**
- **Dialog (i.e. a Properties window)**
- **Graph**
- **Groupbox**
- **HTML**
- **IconGrid**
- **Panel**
- **Table**
- **Tree**

Other Controls Used by FlexSim

- **Script (code editor)**
- **Histogram**
- **Pie Chart**
- **Line Chart, Bar Chart, Scatter Plot**

There are also some controls available in the GUI builder's icon grid that are made up of a combination of one or more of the controls mentioned above. These controls will not be documented in this topic since they are simply a combination of functionality for controls that are documented here.

General Attributes

Below is a list of general attributes that can be added to almost every view type.

alignrightmargin,alignbottommargin: These attributes signal that the control's margin is "locked" to the right or bottom margin of the control's container window. They should container number data, and the value represents the offset distance from the container window's right or bottom margin, in pixels, that the control's margin will be "locked" to. For example, if you give a button control an alignrightmargin attribute with a value of 10, then as you resize the window, the button will automatically resize so that its right margin is 10 pixels from the right edge of the window.

alignrightposition,alignbottomposition: These attributes signal that the control's position is "locked" to the right or bottom margin of the control's container window. They should container number data, and the value represents the offset distance from the container window's right or bottom margin, in pixels, that the control's position will be "locked" to. For example, if you give a button control an alignrightposition attribute with a value of 100, then as you resize the window, the button's x position will automatically change so that its button's position (or left side) is 100 pixels from the right edge of the window.

aligncenterx, aligncentery: These attributes signal that the control should be center aligned with its container control. The attributes should contain number data, and their value represents an offset from the center position.

grayed: This attribute causes the control to be disabled, graying the control and disallowing the user from manipulating the control. It should container number data; 1 means grayed, 0 means not grayed. Once a control has been initialized, the "grayed" state of the control will not change by simply changing the value of the grayed attribute. Because of this, and because the "grayed" state of a control is very dependent on certain parameters of objects, and can change during the life of the window, it is usually more practical to use the windowgray() command to change the "grayed" state of a control. This command does not require you to even have a grayed attribute, so the grayed attribute, for the most part, is unneeded.

hidden: This attribute designates that the control will be hidden from the user. For the same reasons as with the grayed attribute, this attribute is usually unneeded and can be replaced with the windowshow() command.

coldlink: This attribute is used to link a control with a value in the model. For example, an edit control with a coldlink to the max content of a queue will show the max content as a text in the edit control. It is a "cold" link because it is only refreshed when the window is opened, and only applied when the Apply or OK button is pressed. The coldlink attribute should contain text data that is a path to the node that holds the linked data. The path starts at the coldlink node itself. Refer to the Graphical User Interfaces topic for more information on the syntax of this path. You can use the applylinks() command to apply or refresh the coldlink. The first parameter is a node that is the start location for a recursive search. The second parameter is optional. The command will recursively search the window's tree structure and find any coldlinks (and hotlinks) and apply the coldlinks to the object's attributes. If the optional second parameter is 1, then instead of applying the coldlinks to the object, the applylinks() command will refresh the window's values as defined by the object's values. For more information on traversing the tree, see Traversing the Tree.

hotlink: This attribute is used to link a control with a value in the model, just like the coldlink mentioned above. It is a "hot" link because the value shown in the control is continuously refreshed each time the window is repainted. Otherwise the hotlink is exactly the same as the coldlink. For more information on traversing the tree, see Traversing the Tree.

coldlinkname: This attribute is just like the coldlink, except it links with the name of the node specified by the coldlink's path. For more information on traversing the tree, see Traversing the Tree.

hotlinkname: This attribute is just like the hotlink, except it links with the name of the node specified by the coldlink's path. For more information on traversing the tree, see Traversing the Tree.

coldlinkx: This attribute is like a coldlink, except that instead of holding text data with a path to the involved node, the coldlinkx holds flexscript code. The flexscript function should return a reference to the node that the view should link to. If 0 is returned, then nothing will be applied or refreshed for the control. Within the function there are 3 access variables. c is a reference to the control itself. i is a reference to the object focus of the view (the same as node("@>objectfocus+",c)). eventdata is either 1 or 0. If 0, then the coldlinks function is being executed in order to refresh the control according to the object's variable. If 1, then the coldlinkx is being executed in order to apply the value to the object's variable. Please note that when the coldlinkx function is called, the return value, or in other words the reference to the linked node, is not remembered by the window. Each time the control needs to be refreshed or applied, the coldlinkx function is called again. This means that the coldlinkx function can actually be called many times throughout the life of the window. For more information on traversing the tree, see Traversing the Tree.

hotlinkx: This attribute is just like the coldlinkx, except that it is refreshed every time the window is repainted. For more information on traversing the tree, see Traversing the Tree.

menupopup: This attribute is only applicable for FlexSim registered controls. By adding this attribute you can define the menu that will appear when the user right-clicks on the control. The attribute should contain sub-nodes. Each sub-node is a menu option of the popup. The sub-node's name defines what the menu item's text will be. The sub-node should have text data. The text data defines a flexscript function that will be executed when the option is selected. Within the flexscript function the is one access variable, c. c is a reference to the menu option sub-node. Some commands that might be used within the flexscript function are listed below. For detailed information on each command, refer to the command documentation.

ownerobject(c): this returns a reference to the ownerobject of the menu option, or the view node itself.

selectedobject(): this will return a reference to the highlighted (yellow) object of the view. For example, if the view is an ortho view, and the user right-clicks on an object in the ortho view and selects a popup menu option, the flexscript function can access the highlighted object with selectedobject(ownerobject(c)).

Standard views that use this attribute include the ortho and perspective windows, the library icon grid, and the table view in the labels tab of an object's properties window.

Right now there is no known way of dynamically customizing the popup menu based on what the user clicks on.

If there is no menupopup attribute specified for a FlexSim registered control, then the standard FlexSim popup menu will appear.

tooltip: This attribute defines a tip that will pop up when the mouse hovers over the control for a certain amount of time. The attribute should have text data containing the text that should be shown.

style: This attribute is a special attribute that connects directly with windows control styles. When each control is created, it is given a default style, which is a 32-bit field that is passed to Windows where each bit represents a certain flag that affects the control's appearance or behavior. The style attribute can override the default style that FlexSim gives the control. Window styles are documented online on Microsoft Developer Network (MSDN). Go to www.msdn.com and search for Window styles. This will provide list of default window styles. There are also styles specific to each Windows common control. For example, you can search for Button Styles and it will show a list of all the styles you can give a button control.

The style attribute can have any number of sub-nodes. Each sub-node's name defines a windows style for that control, such as WS_DISABLED or WS_BORDER. The sub-node may contain optional number data. If the number data is 0, then that will signal for FlexSim to set the bit flag low, or 0, in case the FlexSim default is for the bit flag to be high. If no number data is specified, then FlexSim will set that flag as a high bit.

An example of using the style attribute is to have a checkbox with the style BS_PUSHBUTTON. This will cause the checkbox to instead look like a button that is depressed when checked. The ortho window's tool bar uses this style for radio buttons in its mode panel.

To create a modal window, use viewwindowtype 4 (Dialog) and add the subnode *FS_MODAL*.

exstyle: This attribute is much like the style attribute, except it defines windows extended styles, which are styles not introduced until Windows 3.1 I believe. These styles begin with WS_EX_ instead of WS_ and are also documented on MSDN.

OnClick: This attribute only applies to FlexSim registered controls. It is fired when the user clicks anywhere inside of the control. This includes when the user clicks the mouse and when the user releases the mouse. There are two access variables. c is a reference to the control node. i is a clickcode: 2 means left mouse button down, 3 mean left up, 4 mean right down, 5 means right up, and 1 means double-click. Some commands that you might use within the OnClick are: cursorinfo(), selectedobject().

OnMouseButtonDown: This attribute only applies to FlexSim registered controls. It is fired when the user presses the left mouse button in the view. There is one access variable, namely c, which is a reference to the control node.

OnMouseButtonUp: This attribute only applies to FlexSim registered controls. It is fired when the user releases the left mouse button in the view. There is one access variable, namely c, which is a reference to the control node.

OnKeyDown: This attribute only applies to FlexSim registered controls. It is fired when the control has keyboard focus and the user presses a key on the keyboard. There is one access variable, namely c, which is a reference to the control node. You can also query what key went down using lastkeydown(), or query whether any key is down with iskeydown().

OnKeyUp: This attribute only applies to FlexSim registered controls. It is fired when the control has keyboard focus and the user presses a key on the keyboard. There is one access variable, namely c, which is a reference to the control node. You can also query what key went down using lastkeydown(), or query whether any key is down with iskeydown().

OnMouseWheel: This attribute only applies to FlexSim registered controls. It is fired when the control has keyboard focus and the user scrolls the mouse wheel. The control also needs to have an OnMouseWheelDelta attribute with number data. When the user scrolls the mouse wheel, FlexSim will set the value of the OnMouseWheelDelta value according to how much the user has scrolled, and then will call the OnMouseWheel function. Within the function, c access the OnMouseWheel attribute itself.

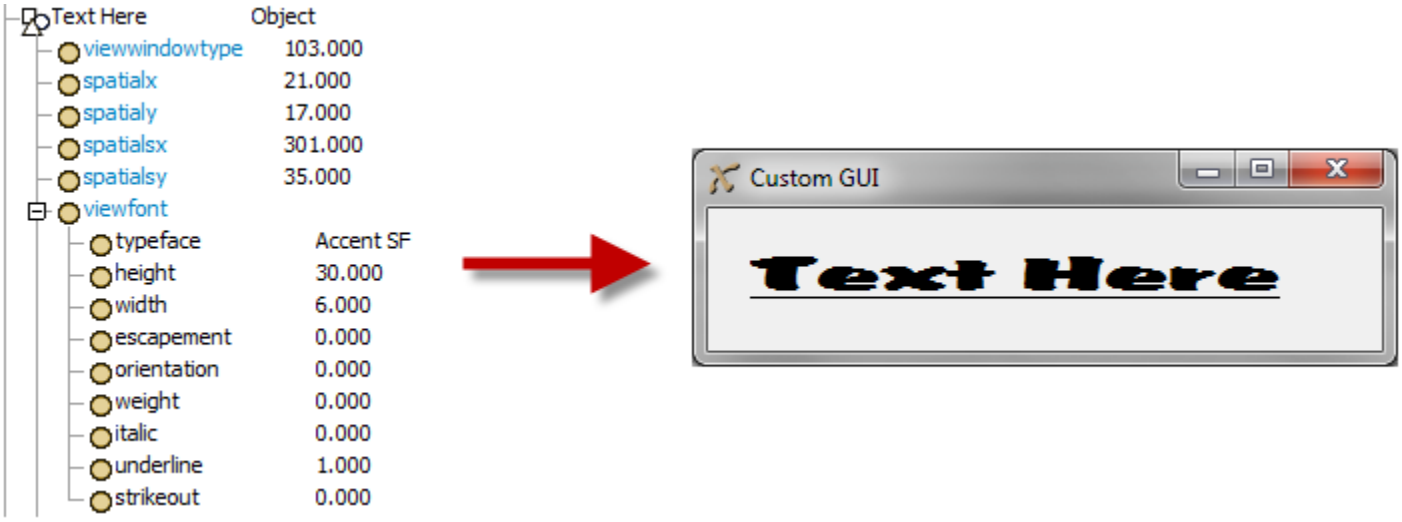
OnMouseWheelDelta: This attribute is used as described in the OnMouseWheel attribute above.

OnDrop: This attribute only applies to FlexSim registered controls. It is fired when the user drags an object from an icon grid and drops it on the view. The attribute should have text data containing flexscript code that fires when the object is dropped. Within the function, you have access to the object that was dragged with dropnodefrom(), and the object that it was dropped onto with dropnodeto().

OnFocus: This attribute is fired when the control receives keyboard focus. You can access the control that just lost focus using nodefromwindow(eventdata).

OnKillFocus: This attribute is fired when the control loses keyboard focus.

viewfont: This attribute only applies to Windows Common Controls. It defines the font that the control will use. The viewfont attribute is a wrapper around the Windows API CreateFont() command. Add this attribute and give it sub-nodes defining the different parameters that will be passed into CreateFont(), as shown in the figure below. You can refer to Microsoft's Windows API documentation for more information on this.



viewbackgroundcolor: For windows common controls, the viewbackgroundcolor attribute defines the text background of the control. For a 3D view, it defines the background color of the scene. The attribute should have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively.

color: For Windows Common Controls, the color attribute defines the text color of the control. The attribute should have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively.

Control Specific Attributes

3D (Ortho/Perspective)

The ortho/perspective view type is a 3D view. The GUI builder's ortho and perspective views include many default attributes that are not documented in detail here, but you can experiment with those attributes from within the GUI builder, as well as look at the standard ortho view and its settings window to see what attributes do what. We document below attributes essential to an understanding of the ortho and perspective views.

viewfocus: This attribute defines what the ortho view is "looking at". It should contain text data with a path to the node containing the objects to be viewed.

viewprojectiontype: The viewprojectiontype attribute specifies whether the view is an ortho or a perspective view. It has number data, 1 signifying ortho, 0 signifying perspective.

viewpointx,viewpointy,viewpointz, viewpointrx, viewpointry, viewpointrz: These attributes specify the focal point and viewing angle of the view.

viewmagnification: This attribute is only used with ortho views, and specifies the magnification or "zoom" of the view.

viewpointradius: This attribute is only used with perspective views, and specifies the camera's distance from the focal point.

OnDropNode: This attribute should contain text data and defines flexscript code that will be executed when the user holds a key down while click-dragging from one object drawn in the view to another.

Button

The button control is a push-able button. Below are attributes that can be used in a button control.

apply: If the apply attribute is added to the button, then FlexSim will call applylinks() on the button's owner view when the button is pressed. The attribute needs no data.

close: If the close attribute is added to the button, then FlexSim will close the button's owner view when the button is pressed. The attribute needs no data.

bitmap: the bitmap attribute specifies a bitmap to be shown on the button. It should contain text data specifying a file path to the bitmap, starting at the FlexSim main directory. If the bitmap attribute is present then the name of the button will not be shown. The file can be a bmp, jpg, png, or ico file.

coldlink, hotlink, coldlinkx, hotlinkx: The link attributes can be added to the button. They work as documented above and designate what will be shown as the button's text.

OnPress: The OnPress attribute specifies code that will be executed when the button is pressed. It should have text data containing flexscript code. Within the code, c accesses the button view.

Checkbox

The checkbox control is a button that has two states, check or unchecked. Below are attributes that can be used with the edit control. A box with a check in it appears, with text to the right of the box. The text of the checkbox is defined by the name of the control node. Below are attributes that can be used with the checkbox control.

coldlink,hotlink,coldlinkx,hotlinkx: The coldlink and hotlink attributes link the state of the checkbox to a value in an attribute of the object. The node that the cold/hotlinks link to should hold number data and have a value of 1 or 0. 1 will cause the box to be checked, 0 will cause the box to be unchecked. You can also explicitly check the box with setchecked() or get whether it is checked with getchecked() (1 means checked, 0 means unchecked).

itemcurrent: The itemcurrent attribute is actually added to the checkbox when an instance is created. Usually it signifies whether the box is checked, but it may not always. Use getchecked() to get the check state of the box.

style: A style that may be useful with the check box is the BS_PUSHLIKE style. There is also a style for having the box be to the right of the text.

Combobox

A combobox is a box containing a dropdown list from which you can select options. Below are listed some of the attributes you can use with comboboxes.

items: The items attribute is a required attribute. It should contain subnodes. Each subnode represents an option in the combobox. The subnode's name specifies the text that will be shown in the option. You can manually add subnodes and give them names if your combobox options do not change. Often you will want the options in the combobox to be dynamic. You can do this by writing code in an OnOpen or the OnPress trigger that populates the items attribute dynamically. Access the items attribute with the items() attribute command, clear the contents of it, then add nodes into the items attribute with nodeinsertinto(), then set their names with setname(). Once you've populated the list, set the itemcurrent attribute's value to the rank of the option that you would like the combobox to reference by default, then call comborefresh() to refresh the combobox windows options according to the new items list.

itemcurrent: The itemcurrent attribute is also a required attribute. It should have number data, and it specifies which attribute is currently selected.

OnSelect: The OnSelect attribute is a trigger that fires when the user selects an option in the combobox. The attribute should have text data with flexscript code specifying what to do when the user selects the option. Within this trigger you will want to access the value of the itemcurrent attribute to find out which option the user selected.

OnPress: The OnPress attribute is a trigger that fires when the user clicks the arrow at the right of the view, right before the combo box displays its list of items. This is a good place to execute code to dynamically change the contents of items.

DateTimePicker

format: This attribute specifies how the DateTimePicker will display the given date or time. This attribute only works if the view has the DTS_APPCANPARSE style. For more information about format options, see the Model Settings page.

coldlink/coldlinkx/hotlink/hotlinkx: This attribute should be a link to a node with number data on it, or return a number respectively. The numerical value is the date/time value that will be displayed. The number should be in unix time, which is the number of seconds since Jan 1st 1601. For example, 13024569600 would be 8:00:00 AM Wed 25 Sep 2013 You can get the current unix time with the command applicationcommand("getunixtime").

OnSelect: The OnSelect attribute lets you execute flexscript code when the user changes the date or time.

OnKillFocus: The OnKillFocus attribute lets you execute flexscript code when the user clicks off of the datetimepicker.

style: The following styles are available to this view: DTS_TIMEFORMAT, DTS_LONGDATEFORMAT, DTS_SHORTDATEFORMAT, DTS_SHORTDATECENTURYFORMAT, DTS_UPDOWN, DTS_APPCANPARSE.

Dialog

The following attributes can be added to a dialog view type, or the view node for the main window.

menucustom: This attribute acts much like the menupopup attribute, except the menu will appear at the top of the window instead of appearing when the user right-clicks in the view.

menuview: This attribute allows you to have the standard FlexSim menu appear at the top of the window. The attribute should have number data and be set to 1 to have the menu appear.

objectfocus: This attribute will specify a path to the object that a given window instance will point to. The createview() command contains a parameter that specifies the objectfocus, the objectfocus attribute will be filled with that parameter when the window is opened.

OnOpen: This attribute allows you to specify flexscript functionality that will fire when the window is opened. It is fired when the window is initially opened, as well as when the window is restored after a compile, as well as when the window is "redirected" to point to a new object if the user switches the window to point to another object. You may use this trigger to initialize settings in controls that may not have a coldlink.

OnPreOpen: This attribute allows you to execute functionality before the window is created. The OnPreOpen is fired after the tree structure for the window is created, but before the window itself is initialize. Unlike OnOpen, it is not executed after a compile or when the window is redirected. You could use the OnPreOpen to modify the structure of the window before it is opened, such as adding or removing tab windows, or adding or removing any controls from the tree structure of the window.

OnClose: This attribute contains text data with flexscript code that will be executed when the window is closed.

palettewindow: This attribute will cause the window's title bar to be smaller and look more like a tool window. The window will also always appear on top of other windows. The attribute needs no data.

windowtitle: This attribute defines the title of the window. It should have text data defining the title.

style: To create a modal window, add the subnode *FS_MODAL* to the style attribute.

Graph

The graph control lets you show data in a graph format. It can be presented as a pie chart, a bar graph, a line graph, a histogram, or a scatter plot. The graph is updated on the fly, so you can have real-time model data be shown as the model runs. Below are attributes that can be used with the graph.

viewfocus: The viewfocus attribute is required. It should have text data containing a path to a node that contains data for the graph. The structure of the data depends on the type of graph. If the graph is a pie chart, then the view focus node should have text data that specifies the name of the chart. If the graph is a line graph, scatter plot, or bar graph, then again the viewfocus node should have text data that specifies the title of the chart, and the viewfocus should have sub-nodes. Every odd-numbered sub-node will represent a point on the x axis, and every even-numbered sub-node represent its corresponding y value. If the graph is a histogram, then the viewfocus node should have three sub-nodes, each having number data. The first sub-node should be the minimum value of the histogram range. The second sub-node should be the maximum value of the histogram range. The third sub-node should be the number of "buckets" or divisions in the histogram.

graphgrid: If this attribute is present, then a grid will be shown as a background for the graph. The width of the grid does not correlate to any set value range in the graph, but is rather for comparative purposes, for example to compare the height of two bars in the graph. The attribute does not need any data.

graphtitle: If this attribute is present, then the grid will show a title for the graph. This is usually the text data on the viewfocus node except in the case of a histogram, where it is the text data on the graphhistodata focus. The attribute does not need any data.

graphaxes: If this attribute is present, then the grid will show the min and max values of the x and y range for the graph. The attribute does not need any data.

Groupbox

The groupbox is a control that groups several other controls together with a heading an a border around it. There are no special attributes for the groupbox.

Edit

The edit control shows text that can be edited by the user. If no special attribtes are given to the control, then it will show the name of the control as its text. Below are attributes that can be used with the edit control.

coldlink,hotlink,coldlinkx,hotlinkx,coldlinkname,hotlinkname: The coldlink and hotlink attributes are most common with this type of control. They connect the text of the edit control with the value of an attribute or variable node of the object. They work as documented above and define what will appear in the text of the control. If the linked node contains number data, then the value will be copied into the control's text with the precision that the user specified in the main Edit menu. When the link is applied, the value in the edit control's text field is copied back to the linked node. You can also use setviewtext() and getviewtext() to explicitly get and set the text of the control.

style: One style that may by useful for the edit control is the ES_READONLY style. This causes the edit's text area to be gray and uneditable, but with the depressed border unlike a static, signifying a value that is feedback to the user but not an input from the user. ES_NUMBER will cause the field to only accept numbers and gives an alert if the user tries to enter a character.

Histogram Attributes

To create a histogram, you should add the following attributes. For an example of the structure needed for a histogram, view the tree of any FlexSim object and go to the tree at >stats/staytime/stats_staytimehisto.

graphhistodata: If this attribute is present, then the graph will show a histogram. The attribute should have text data containing a path to a node that contains "bucket" sub-nodes. The number of sub-nodes should be the number specified by the viewfocus plus 2. The first sub-node will be designated as the "underflow" node, where any values less than the minimum range value of the histogram will be added to this node. The last node is "overflow" for values that are greater than the histogram's maximum range. Every other sub-node represents the histogram value for the "bucket" for the sub-node's corresponding interval.

graphhistointervaldata: You can also allow FlexSim to calculate a confidence interval on the mean of the histogram data by adding this attribute. This attribute should contain text data with a path to a node with 5 sub-nodes. The node itself should have number data that is either 1 or 0. If 1, the confidence interval will be shown on the graph. The first three sub-nodes are used by FlexSim and should have number data. The 4th sub-node should be the percent confidence, and the 5th sub-node should have number data with the value 1, telling FlexSim to automatically calculate the interval.

graphbars: This attribute signifies that the graph will be shown as bars. The attribute needs no data.

HTML

The HTML window is used in GUIs such as the Start Page, Online Content Page, and Dashboards.

html: The html node in the view's variables node will be the html that is displayed on the view.

OnPreLoad: The OnPreLoad node in the view's eventfunctions node (must be toggle as Flexscript) will be executed prior to the html being loaded on the page.

Icongrid

The icongrid control provides a drag-drop mechanism for the user. The library icon grid is an icongrid control. The GUI builder also uses icongrids to drag-drop controls and attributes into the GUI. Below are attributes that can be used with the icongrid.

viewfocus: The viewfocus attribute is a required attribute. It should have text data that contains a path to the view focus of the icon grid. The icongrid will display each subnode of the view focus as a draggable rectangle. In order for a given sub-node to be displayed, it must have object data, and it must have a picture attribute. If not, the object will not be shown at all in the icongrid. The picture attribute of the object may contain text data that specifies a path to a bitmap file, starting in the FlexSim main directory. The file can be a bmp, jpg, png, or ico file. If the path is valid, then the picture will be shown in the object's rectangle. Otherwise, only the name of the object will be shown.

viewwindowsource: This attribute is required if you want the icongrid to be draggable. It should have number data with the value of 1, meaning yes you want the user to be able to drag objects from the icongrid onto other views.

cellwidth: This attribute allows you to define the width, in pixels, of each rectangle in the icongrid. It should have number data containing the desired width.

cellheight: This attribute allows you to define the height, in pixels, of each rectangle in the icongrid. It should have number data containing the desired height.

displaygroup: This attribute lets you display just a sub-group of the view focus. The attribute should have text data specifying a name for the sub-group, like "Manufacturing". Then, each object in the view focus should also have a displaygroup attribute with text. If the displaygroup of the object matches the displaygroup of the icongrid, then the object will be shown in the icongrid. To change the group that you want shown, just change the text of the incongrid's displaygroup attribute, then call repaintview() on the icongrid.

viewhidealllabels: This attribute causes the objects' names not to be shown, but only their pictures. The attribute should have number data with a value of 1.

viewbackgroundcolor: If this attribute is present, then the icongrid will not be shown as a set of raised buttons, but instead will just paint the pictures and names on top of a background you specify. The attribute should have 3 sub-nodes, each with number data between 0 and 1 corresponding to the red, green and blue components respectively.

depresshighlighted: If this attribute is present, then the view's highlighted object will have a sunken border instead of a raised border. The attribute does not need any data. To access the view's highlighted object, use the selectedobject() command.

picturealignleft: If this attribute is present, then each object's icon will be shown to the left of the object's name, instead of above it and in the center. The attribute should have number data specifying the width, in pixels, to make available to the left of the object's name for the object's picture.

OnDrag: This attribute allows you to execute code when an object is dragged from the icongrid onto another view. The attribute should have text data with flexscript code. Within the function, c gets access to the icongrid, i gets access to the view on which it was dropped, dropx(),dropy(), and dropz() get the drop location if the view is a ortho, perspective, dropnodefrom() gets access to the object that was dragged, and dropnodeto() gets access to the object it was dropped onto if one exists. Please note that if no OnDrag attribute exists, then a copy of the object will be made in the dropped view's focus or in dropnodeto() if it exists.

Line Chart, Bar Chart, Scatter Plot Attributes

If the graph is not a pie chart or histogram (and it has a graphxy attribute), then the sub-nodes of the focus node will be interpreted as paired values to be plotted on the x and y axes of the chart. Odd-numbered sub-nodes represent x values and even-numbered sub-nodes represent y values. Use the following attributes to define how those values will be drawn. The attributes can be combined as needed.

graphxy: Add this attribute to make each sub-node pair be interpreted as an x/y value. If this attribute is not present, then each sub-node (not sub-node pair) will be interpreted as a y value, and the points will be evenly distributed across the x-axis, the first point with the x value 1, the second with x-value 2, and so

forth. You may want to leave this attribute out if you want to display a bar chart where the bars are evenly distributed across the x axis.

graphlines: Add this attribute to have a line drawn between consecutive points in the graph. The attribute needs no data.

graphpoints: Add this attribute to have a dot drawn at each point in the graph. The attribute needs no data.

graphbars: Add this attribute to have a bar drawn from 0 to the y value at each point in the graph. The attribute needs no data.

graphstep: If this attribute and the graphlines attribute are present, then each line will be drawn first horizontally and then vertically to the next point, creating a stair-step effect instead of diagonal lines.

gridx, gridy: If the graphgrid attribute is present, then you can also add a gridx and/or gridy attribute, which specifies the x/y grid interval.

graphscatterdata: This attributes causes the structure of the data to be redefined. The attribute should have two sub-nodes, each containing text data defining a path to a container node. The first sub-node's focus node contains a list of "x" data points, and the second sub-node's focus node contains a list of "y" data points. Each point on the graph consists of an x-value from the first focus node paired with a y-value from the second focus node. If this attribute is present, then the viewfocus attribute will be ignored.

dataseries: The dataseries node is actually not an attribute on the graph control. Instead, it should be placed in the same container as the focus node (>viewfocus+/../). If the graph control sees that this node is present, then it will split the graph up into several uniquely colored data series. This allows you to have a line graph with multiple colored lines, or a bar graph with multiple colors and a legend. The dataseries node should have sub-nodes, each representing one data series. The name of each sub-node will be shown in the graph's legend. Each sub-node should also contain number data that signifies how many points are in that data series. The value of the sub-node specifies an "up-to" point in the data. Take for example a line chart with 20 points. Normally the graph will simply draw a red line between each point in the graph. However, let's say you want to split the 20 points into 3 categories. The first 7 points represent the content graph for Object A, the next 10 points represent the content graph for Object B, and the last 3 points represent the content graph for Object C. To split these data series, add a node called dataseries to the content graph's container node. Then add three sub-nodes, named Object A, Object B, and Object C. Then give each sub-node the respective values: 7, 17, 20. This designates Object A's series as points 1-7, Object B's as points 8-17, and Object C's as points 18-20. The graph should now show three uniquely colored lines, as well as a legend.

Listbox

The listbox is similar to the combobox, except the options are not displayed in a window that drops down. Instead, then list is displayed in the window itself. Below are attributes that can be used with the listbox control.

items: The items attribute is required on the listbox. It represents the list of options in the list box. It acts just like the items attribute in the combobox.

itemcurrent: The itemcurrent attribute is also required. It also acts just like the itemcurrent attribute of the combobox.

OnSelect: The OnSelect attribute is a trigger. It acts just like the OnSelect attribute of the combobox.

Panel

The panel control is like a groupbox but with a different border. You can give the panel a sunken border or no border at all. Below are attributes that can be used with the panel control.

beveltype: the beveltype attribute specifies what the border of the panel should look like. The attribute should have number data with a value between 0 and 2. A value of 0 will cause no border to be drawn. A value of 1 causes a one pixel sunken border to be drawn. A value of 2 causes a 2 pixel border to be drawn.

bitmap: You can give the panel a bitmap. To do this, add the bitmap attribute, give it the path to the bitmap file (like buttons\up_arrow.bmp), then give the panel a viewfocus attribute with the following text: "..>bitmap". This will cause the bitmap to be shown on the panel. The file can be a bmp, jpg, png, or ico file.

color: You can also have the panel show a certain color. To do this, give the panel a color attribute as well as a viewfocus attribute. If the color you want to display resides on the object itself, then the text of the viewfocus should be something like: "@>objectfocus+>visual/color". If you would like to store the color on the view itself, then give the color attribute three subnodes, each with a number value between 0 and 1 representing the red, green and blue values respectively.

splitterx, splittery: The splitterx and splittery attributes designate the panel as a container for two resizable subcontrols. You should use one or the other but not both attributes. The attribute does not need data. To set up a splitter panel, add then the attribute, then add two subcontrols to the panel by dropping them inside the panel. The way the panel works is, if the mouse clicks and drags on any portion of the panel that is not part of a subcontrol, then the subcontrols will be resized. Thus you want to have the sub-controls take up the entire area of the panel except for a small "resizer" bar in the middle of the panel. As an example, drag a panel into your GUI and then give it a size of 200 x 200. Then drag two buttons into the panel. Give the first button a location of (0,0) and a size of (100,200). Give the second button a location of (105, 0) and a size of (95,200). Then give the panel a splitterx attribute. Then press F5 to go into preview mode. Notice the 5 pixel wide bar in the middle. Click it and drag the mouse right or left. Notice that the buttons will be resized as you drag. This example is not a very practical example, but the panel can be used with any controls. For a more useful example, look at the picklist code editor, which uses a splittery panel to show the template code and actual code simultaneously (VIEW:/standardviews/picklistedit).

Pie Chart Attributes

To create a pie chart, add the following attributes.

graphpie: This attribute signifies that the graph will be shown as a pie chart. The attribute needs no data.

graphpiedata: This attribute should have text data the contains a path to a node with sub-nodes. Each sub-node's name will be show in the pie chart's legend, and should have number data that is the value for that item in the pie chart. The sum of all sub-nodes of the focus node will represent a full 360 degree circle, and the each sub-node's value represents a chunk of the pie.

Radiobutton

The radiobutton control is much like a check box in that it can have two states, but it also has some extra functionality where only one radiobutton can be checked at one time within a container of radiobuttons. It allows the user to choose one of several options. Below are attributes that can be used with the radiobutton control.

coldlink,hotlink,coldlinkx,hotlinkx: The coldlink and hotlink attributes link the state of the radiobutton to a value in an attribute of the object. The node that the cold/hotlinks link to should hold number data and have a value of 1 or 0. 1 will cause the button to be checked, 0 will cause the button to be unchecked. You can also explicitly check the box with setchecked() or get whether it is checked with getchecked() (1 means checked, 0 means unchecked). Radio buttons present a problem because in order to have them work using just cold/hotlinks, you would need to connect each one to an individual node's value. Hence you would need 5 nodes in the object's attributes with only one having the value of 1 and the rest having 0. What users often want instead of this is to have a set of radiobuttons that reflect one value. For example, if a node has value 1, then radio button 1 should be checked; for value 2, radio button 2 should be check, and so forth. In order to do this you would need to write your own code in a coldlinkx or in the OnOpen of the window. Alternatively, you can use a combobox control. The combobox control fulfills the same functional requirement of choosing one of several options, but packages it all into one control that can be linked to a value on one node.

itemcurrent: Again, the itemcurrent attribute is added to the radiobutton automatically when an instance is created. Usually it signifies whether the box is checked, but it may not always. Use getchecked() to get the check state of the box.

style: Again, it may be useful to use the BS_PUSHLIKE style here.

Script

The script control uses the Scintilla text editor. This is a flexible code editor that provides FlexSim with many code editing features. To learn more about the Scintilla text editor, go to www.scintilla.org. Below are attributes that you can add to a script control.

coldlink: The coldlink attribute links the text of the script control to the text of a node on the object. It works as documented in the general section above.

noformat: The noformat attribute causes the script control to be unformatted, meaning no code highlighting, line numbers, or folding will be done to the text of the control.

You can explicitly get the text of the control with getviewtext(), and set it with setviewtext(). You can also use a script control to display template code using the codetotemplate() command, which fills the template text of the control based on the code of a node, and the templatetocode() command, which modifies the code on a node based on the changes that the user has made to the template text. If you use the script control to show template code, you should give the control a noformat attribute.

Scrollbar

OnDrag: This attribute allows you to execute code when the scroll bar is dragged.

vertical: If this attribute is added, the scrollbar will be a vertical scrollbar. This attribute needs no data.

Spincontrol

viewfocus: This attribute defines what the field the spinner is tied to.

OnClick: This attribute is a Flexscript toggled node in the eventfunctions node. It is called when the user mouse downs and mouse ups on the control.

OnMouseMove: This attribute is a Flexscript toggled node in the eventfunctions node. It is called after the user mouses down on the control, and before they mouse up.

Static

The static (or label) control is a control that simply shows text. If no special attributes are given to the control, then the name of the control will be shown as its text. Below are attributes that can be used with the static control.

coldlink,hotlink,coldlinkx,hotlinkx,coldlinkname,hotlinkname: The coldlink and hotlink attributes allow you to have the text be dynamic based and attribute of the object. They work as documented above and define the text of the static control. Because the text of the static control is not editable, the links are not applied as they would be for other controls. You can also use setviewtext() and getviewtext() to explicitly get and set the text of the control.

bitmap: The bitmap attribute causes the static control to show a bitmap instead of text. The attribute should have text data that defines a path to the bitmap file, starting at the FlexSim main directory. The file can be a bmp, jpg, png, or ico file. If the file is a bmp, you can specify within the bitmap file certain areas a "transparent", meaning the standard background color of the view will show through. To do this, the bitmap must be created in index color mode 24 bits per pixel, and the color that Windows will designate as transparent is the color R:192 G:192 B:192.

Statusbar

The statusbar control adds a status bar to the bottom of another window. You can set the text of the statusbar with setviewtext(). Below are attributes you can use with the status bar:

coldlink, hotlink, coldlinkx, hotlinkx: These attributes link the text of the status bar to the value on a node. They work as documented above.

Tabcontrol

The tabcontrol is a control that contains sub-controls that are each a tab page. To add a tab page, drag a control into the tab. Below are attributes that can be used with the tabcontrol.

pagelist: The pagelist attribute allows you to have an external source for the pages of the tab control. The attribute should have text data specifying a path to a node that contains subnodes. Each of the subnodes should contain text data that is a path to the page control. The pagelist attribute is used quite often in object Properties windows.

itemcurrent: The itemcurrent attribute is required. It allows you to know which tab is currently selected. It should contain number data whose value will be set whenever the user chooses a tab.

OnSelect: The OnSelect attribute lets you execute flexscript code when a tab page is selected. The attribute should have text data that will be fired when the user selects a tab. Within the code, c accesses the tabcontrol view. Use get(itemcurrent(c)) to get the currently selected page.

Table

The table control is a window that allows the user to see and edit a table or list of data. Below are attributes you can use with the table control.

viewfocus: The viewfocus attribute is required for the table control. It specifies what the table will "point at". It is like the objectfocus attribute. It should have text data that is a path to the node that contains the table information. There are two options for the structure of the table node itself. It can be a list, meaning it contains a number of subnodes. If the table node is a list, then the table control will show just one column of data, and the text in each entry is the data, either text or number, of each of the subnodes in the list. The row headers of the table are defined by the names of each subnode. The other option is to have the table node be an actual table. In this case the table node contains a set of subnodes. Each subnode is one row in the table, and it contains subnodes, each being an individual entry in the table. The number of columns that will be shown in the table is defined by the number of subnodes of the first row. Row headers again are defined by the names of each row node. Column headers are defined by the names of the subnodes of the first row node. Please note that with a table, it is a direct view into the table data. This means that any changes made in an entry in the table will be applied as soon as you click off of the entry. This is different than using coldlink in other controls, where changes are only made when the user hits the Apply button. This is why the table control uses the viewfocus attribute, because it is a direct view into the data of a model, just like an ortho or perspective control.

dataentry: If you add a dataentry attribute to a table control, then this will allow the user to quickly traverse the table entries using the Tab and Enter keys as well as the arrow keys. This attribute will usually be used. The only time you may not want it is if you may want the user's Tab, Enter and arrow keystrokes to be captured within the entry itself, for example if a table entry may contain multi-line code.

noformat: Like the script control, the table control uses the Scintilla text editor. By adding the noformat attribute, Scintilla features like code highlighting, line numbers, folding, etc. are disabled. Usually this attribute should be present as it makes quick table editing easier.

cellwidth: This attribute specifies the default column width, in pixels, of the table. The attribute should have number data with the value in pixels. The cellwidth attribute can also contain subnodes. Each subnode should have number data that defines the column width of an individual column. The first subnode defines the column width of the row header column, the second subnode defines the first column's width, and so forth.

cellheight: This attribute specifies the height, in pixels, of each row in the table. Unlike table columns, all table rows must be the same.

drawlines: The drawlines attribute allows you to customize how lines between columns and rows are drawn. The attribute should have number data with a value between 0 and 3. A value of 1 will cause no separation lines to be drawn at all. A value of 1 causes both column and row separation lines to be drawn. A value of 2 causes only column separation lines to be drawn. A value of 3 causes only row separation lines to be drawn.

list: The list attribute designates the table as a list of node names. The attribute needs no data. The table will have only one column (no row headers column). In the single column, only the names of each subnode of the table will be shown.

Trackbar

The trackbar control is a control that allows you set numeric values by visually dragging a locator along a trackbar. This control is also known as a slider. Below are attributes that can be used with the trackbar.

coldlink,hotlink,coldlinkx,hotlinkx: These attributes link the tracker to a value on a node. They work as documented in the general section above.

itemcurrent: The itemcurrent attribute is a required attribute. Its value holds the current value that has been selected for the tracker.

rangemin, rangemax: The rangemin and rangemax attributes specify the min and max values of the tracker. They should have number data specifying the min and max values respectively.

rangeexp: This attribute tells the tracker to exponentially increase the value as the locator is dragged to the right, instead of linearly. It should have number data specifying the factor to exponentially increase by. The run speed control at the bottom of the FlexSim window uses this attribute.

OnSelect: This attribute allows you to execute code when the user drags the locator to a given position. The attribute should have text data with flexscript code specifying what to do. Within the function, c will access the tracker control. To get the currently selected value, access the value in the itemcurrent attribute.

Tree

The tree view into flexsim's tree. Below are attributes that can be used with the tree control. For the most part, necessary attributes are added automatically for you. This will document only the attributes that you will need to know about.

viewfocus: The viewfocus specifies the focus of the tree view. It should have text data with a path to the node that will be the head of the tree.

cellwidth: The cellwidth attribute lets you specify how wide, in pixels, each node's name will be shown.

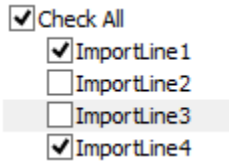
noformat: If this attribute is present, then when text data is edited it will not be formatted for code.

viewpointx: This attribute is required and has number data specifying the x viewpoint, in pixels, of the tree view.

viewpointy: This attribute is required and has number data specifying the y viewpoint, in pixels, of the tree view.

Treeview

This control is not to be mistkan for the FlexSim Tree. The treeview control can be seen in such GUIs as the Excel Interface and the Breakpoints window.



coldlinkx,hotlinkx: These attributes can be used to execute code to update the treeview.

items: The items attribute is required for this control. It contains all of the items that will be displayed in the control. IT is important that each item has number data. The check boxes set and get their state based on the number in each item (1 or 0). Items may have subnodes which will create a parent/child display in the treeview.

itemcurrent: The itemcurrent attribute is required for this control. It contains the currently selected item.

OnPress: The OnPress trigger is called when the user clicks on a check box.

OnSelect: The OnSelect trigger is called when the user selects one of the items in the tree.

style: Styles that are important with this control are TVS_CHECKBOXES, TVS_SHOWSELALWAYS, TVS_FULLROWSELECT, TVS_DISABLEDRAGDROP, TVS_EDITLABELS, TVS_HASBUTTONS, TVS_HASLINES, TVS_LINESATROOT, TVS_SINGLEEXPAND.

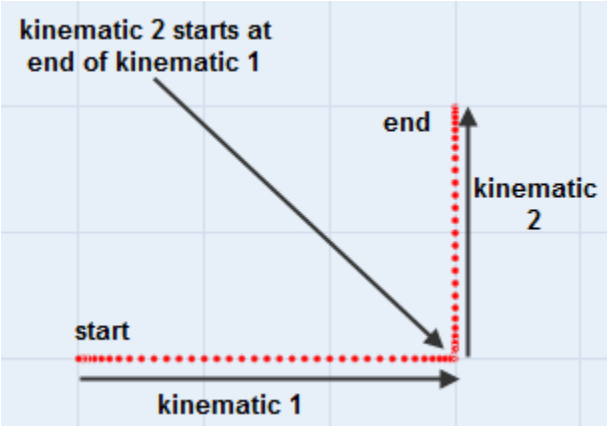
Kinematics

1. Concepts
2. Commands
3. Tutorial

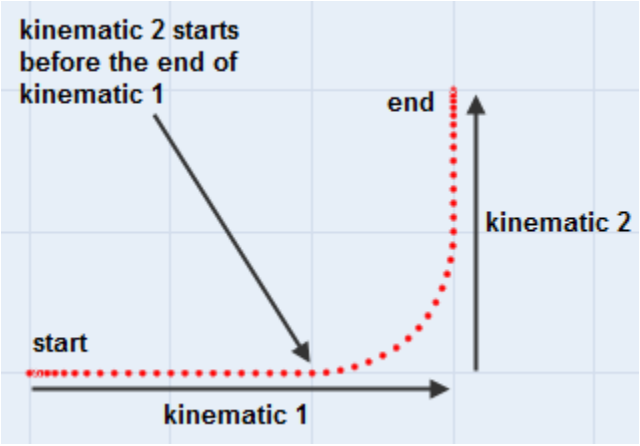
Kinematics Concepts

FlexSim's kinematics functionality allows you to have a single object perform several travel operations through one common interface. Each travel operation can have its own acceleration, deceleration, startspeed, endspeed, and maximum speed properties. Travel operations can overlap with each other, or be performed in sequence.

Below is a time-based plot of two kinematics performed in sequence:



And a time-based plot of two kinematics that overlap each other:



Kinematics can also be used just to make it easier to handle the math for any logic that involves speeds, rates, accelerations, decelerations, etc.

As an example, an overhead crane usually has several motors that drive it. One motor drives the bridge along a railing, while another motor drives a trolley along the bridge, while another lifts the hook or grabber by a cable. Each of these motors may have their own acceleration, deceleration, and maximum speed properties. Using kinematics, you can define all of these motions through a single kinematics interface, where different motors can work simultaneously, giving the motion of the crane's end effector a very dynamic behavior. Before kinematics were introduced, the simplest way to simulate this behavior was to have three different objects hierarchically ordered in the model tree, each object simulating one motion or kinematic. While this works very well in some cases, in other instances it can become tedious and unfriendly. Kinematic functionality attempts to fix this problem by allowing one object to do several motions or kinematics simultaneously.

Commands

1. **initkinematics -** This initializes data for the kinematics, saving things like the start location and rotation of the object you want to apply motion to.
2. **addkinematic -** This gives travel/rotate operations to the object. For example, you can tell the object, starting in 5 seconds and travel 10 units in the x direction, with a given acceleration, deceleration, and

max speed. Then you can tell the object, starting in 7 seconds, travel 10 units in the y direction, with a different acceleration, deceleration, and max speed. The effect of these two operations is that the object will start traveling in the x, then will start simultaneously accelerating in the y, following a parabolic curved path to the destination. Each call to addkinematic will add another operation to the object.

3. **updatekinematics -** This command should be called when the object is being redrawn. This calculates the current position and rotation of the kinematics based on the current time, and sets the object's location to that position.

Refer to the Commands page for information about kinematics commands.

Kinematics Commands

Commands

- **initkinematics**
- **addkinematic**
- **updatekinematics**
- **Other Kinematics Commands**

initkinematics

The Blank Node

For the initkinematics command, as well as all other kinematics commands, you must pass a reference to a blank node as the first parameter. This specifies where you want kinematic information to be stored, or, if you are getting information out of it, where kinematic information has been stored. The node needs to be an otherwise unused node, so that kinematic functionality can store data as needed. For modelers, this node will most likely be a label. Once kinematics have been initialized, the node will display the text "do not touch". This node should not be clicked on in a tree or table view while a kinematic operation is in process, or the kinematic data may be corrupted.

Overloading

The initkinematics command is overloaded, so you can call initkinematics with two different parameter sets, depending on your situation. Parameters for these two overloads are as follows.

void initkinematics(treenode datanode, treenode object [, int flags])

datanode - This is the blank node for kinematic data.

object - This is the object that will do the moving.

flags - This optional parameter defines various flags for the kinematic. It can be a bitwise or on the following values:

KINEMATIC_MANAGE_ROTATIONS - If this flag is set, the object will always point in the direction of travel.
KINEMATIC_RELATIVE_LOCS - If this flag is set, then the object will travel according to its local coordinate system, based on the rotation of the object at the time you initialize the kinematic. If not, kinematic coordinates are based on the object's container's coordinate system.
KINEMATIC_DO_NOT_PRUNE - By default, the kinematics logic will "prune" the kinematics as they go, meaning when you update the kinematics to a time that is after the end time of a given kinematic then it will remove that kinematic from the kinematics list. If you do not want it to do this, i.e. you may be doing queries across various times in the kinematic, then you should set this flag.
KINEMATIC_RESET_BUFFER - By default, the kinematic's memory allocation is optimized for speed, i.e. it will only reallocate memory when it needs it, and it will not resize back down when you initialize the kinematic. However, if this flag is set, it will resize its data buffer to an initial size when initialized.

This parameter set assumes you have an object that you want to move, like a transporter. The first parameter, again, is a blank node kinematic data, either a label, attribute, or variable. The second parameter is the object that will do the moving. The command will save off the object's initial location and rotation. The optional parameter managerotation is either 1 or 0. If 1, the rotation of the object will be set according to the velocity of the object at any given time. By default, the object's positive x direction will always point in the direction of the object's current velocity. This would be used, for example, if you have a truck that you always want to point forward as it travels. If managerotation is passed as 0, then the object won't rotate unless given commands to rotate, which will be explained later. If you do not specify this parameter, then by default, managerotate is set to 0. The optional localcoords parameter specifies the orientation of subsequent travel command locations. For example, if my truck is rotated 45 degrees, and I want it to travel 5 units in the x direction, this can be interpreted in two different ways. Do I want it to travel 5 units in x according to the truck's coordinate system, or 5 units in x according to the model's coordinate

system (or the coordinate system of the truck's container)? In the former case, the object would actually travel 3.5 units in the x direction and 3.5 units in the y direction according to the model's coordinate space. In the latter case, however, the object would travel the usual 5 in the x direction according to the model's coordinate space. The localcoords parameter specifies which coordinate system you want to use. If 1 is passed, the object's coordinate system will be used (the former case). Note that only the object's initial coordinate system will be used to calculate locations, not subsequent coordinate systems if the object rotates later on in the kinematics. If 0 is passed, the object's container's coordinate system will be used (the latter case). If you do not specify this parameter, the default is 0.

void initkinematics(treenode datanode [, double x, double y, double z, double rx, double ry, double rz, int flags])

datanode - This is the blank node for kinematic data.

x, y, z - These are optional initial location variables.
Defaults: 0, 0, 0

rx, ry, rz - These are optional initial rotation variables.
Defaults: 0, 0, 0

flags - See the first overload above for more information.

This parameter set allows you to explicitly pass initial locations and rotations, instead of referencing an object. Although you would probably more often use the other parameter set where you pass the object in, this parameter set gives you ultimate flexibility. Use this if you want to explicitly pass in the initial locations and rotations of the object, or if your location and rotation values don't necessarily represent real locations and rotations in your model. For example, you are simulating a robot arm, and there are several joints of the arm that move/rotation with different acceleration/deceleration/max speed values. The visualization of movements of the arm are not simulated with explicit Flexsim locations/rotations, but are done using your own draw commands and labels or variables. In this case, you don't want kinematics to be applied to straight rotations and locations, but rather to information that you are keeping on the object yourself. In such situations, a given set of kinematics does not need to be viewed as applied directly to x,y,z locations and x,y,z, rotations, but can rather be viewed as six separate kinematic motions, each along one axis. These six axes can represent whatever you want them to. For example, your robot has four joints, each with one rotation value. To have the four joints of the robot move using kinematics, you can have each joint simulate one axis in the kinematics. The x part of the kinematics applies to the rotation of joint 1, the y part applies to the rotation of joint 2, the z part applies to the rotation of joint 3, and the rx part applies to the rotation of joint 4. The other two parts of the kinematics, ry and rz, you don't worry about. You can initialize the kinematics with the start rotations of each of your 4 joints, and then add kinematics that apply to each joint individually. Then, when you want to get the joints' current rotation values out later to draw the robot arm in motion, you can use the getkinematics command instead of the updatekinematics command to get the values explicitly, and not have them be applied to an object's location or rotation. These commands will be explained later.

void setkinematicsrotoffset(treenode datanode, double rx, double ry, double rz)

datanode - This is the blank node for kinematic data.

rx, ry, rz - These are initial rotation variables.

This command would only be used if managerotation is passed into initkinematics as 1. This allows you to set an initial rotation from which the rotation is managed. By default, the object's positive x direction will always point in the direction of the object's current velocity. In the case of a truck, you may want the truck, instead of always being rotated so that it is traveling forward, to travel backward. Here you could specify a rotation offset of (0,0,180).

addkinematic

double addkinematic(treenode datanode, double x, double y, double z, double targetspeed [, double acc, double dec, double startspeed, double endspeed, double starttime, int type])

datanode - This is the blank node for kinematic data.

x, y, z - These are offset locations or rotations.

targetspeed This is the target travel speed.

acc - This optional parameter specifies the acceleration.
Default: 0 (or infinite acceleration)

dec - This optional parameter specifies the deceleration.
Default: 0 (or infinite deceleration)

startspeed - This optional parameter specifies the initial velocity.
Default: 0

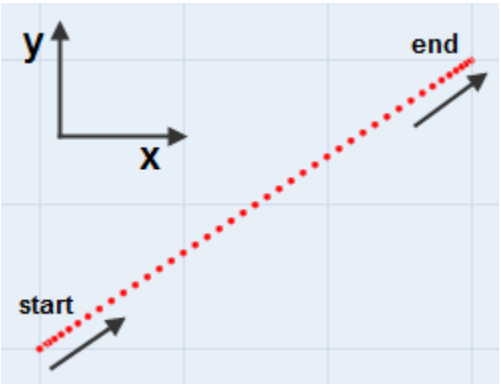
endspeed - This optional parameter specifies the ending velocity.
Default: 0

starttime - This optional parameter specifies the absolute start time.
Default: Current Time

type - This optional parameter specifies the type of travel to apply (rotation or travel).
Default: KINEMATIC_TRAVEL

This command adds a kinematic to the set of kinematics. The x, y, and z parameters make up an offset location or rotation. For example, the location (5,5,0) tells the kinematic to travel 5 in the x and 5 in the y. Note that these are offsets from the object's current position, and not absolute locations. The targetspeed parameter specifies the target speed for the travel operation. The other parameters are optional. Acc specifies the acceleration. Dec specifies the deceleration. Startspeed specifies the speed that the kinematic should start at. If this speed is higher that the target speed, then the object will start at the start speed and decelerate down to the target speed. Endspeed specifies the ending speed for the operation. If endspeed is greater than targetspeed, then at the end of the operation, the object will accelerate from the target speed to the end speed. The starttime is the start time of the kinematic, in simulation time, not as an offset from the current time. The type parameter specifies what type of kinematic it is. This value will usually either be KINEMATIC_TRAVEL, or KINEMATIC_ROTATE. If it is KINEMATIC_TRAVEL, the operation will be applied to the x, y, and z location values. If it is KINEMATIC_ROTATE, the operation will be applied to the rx, ry, and rz rotation values, and speeds are defined in degrees per unit of time, accelerations/decelerations in degrees per unit of time squared. The command returns the time that this kinematic operation will finish.

Below is a time-based plot of a kinematic that travels 3 in the x and 2 in the y, with accelerations and decelerations.



Turn Kinematic Types

In addition to KINEMATIC_TRAVEL and KINEMATIC_ROTATE, there are three "turn" kinematic types, which cause the moving object to turn a certain angle with a given turn radius. These are

KINEMATIC_TURN_XY, KINEMATIC_TURN_YZ, and KINEMATIC_TURN_ZX. For these kinematic types, the x, y, and z parameters that you pass into addkinematic() take on a different meaning, as follows:

x - The start angle in degrees. An angle of 0 means the initial direction of motion when the turn starts will be in the same direction as the first axis of the kinematic type's name, i.e. start angle 0 for KINEMATIC_TURN_XY means the initial motion is positive along the x axis, where for KINEMATIC_TURN_YZ the initial motion would be positive along the y axis. Use the right hand rule around a given axis when figuring out start and turn angles.

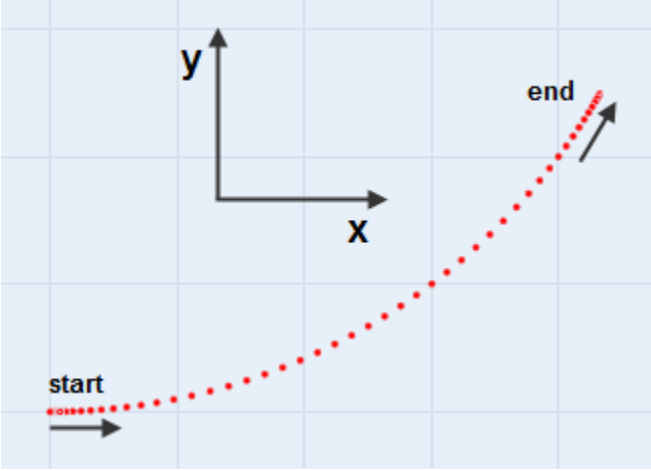
y - The turn angle in degrees. The motion of the object will turn this number of degrees, parallel to the plane defined by the two axes in the kinematic type's name, i.e. KINEMATIC_TURN_XY will turn in the x/y plane, i.e. around the z axis.

z - The turn radius in distance.

All other parameters are the same for this kinematic type, i.e. max speed, acceleration, deceleration, etc. Speeds are in length units per time unit, NOT in degrees per time unit.

As an example, if you would like to make a motion that turns from travelling positive along the x axis to travelling at 60 degrees off the x axis, with a turn radius of 5 (i.e. turn about the z axis), call:

addkinematic(kinematics, 0, 60, 5, 1, 1, 1, 0, 0, time(), KINEMATIC_TURN_XY);



updatekinematics

void updatekinematics(treenode datanode, treenode object [, double updatetime])

datanode - This is the blank node for kinematic data.

object - This is the object that will do the moving.

updatetime - This optional parameter specifies the absolute update time.
Default: Current Time

This command should be called in the middle of the kinematic operation, usually on predraw or draw. It calculates and then sets the current location and rotation of the object, according to all kinematics that have been added and the current update time. The updatetime parameter is optional. If it is not passed, the current simulation time is used.

Other Kinematics Commands

double getkinematics(treenode datanode, int type [, int kinematicindex, double updatetime/traveldist])

datanode - This is the blank node for kinematic data.

type - This specifies the type of information to retrieve (listed below).

kinematicindex - This optional parameter specifies which kinematic to query.
Default: 0 (all kinematics)

updatetme/traveldist - This optional parameter represents either the update time or the travel distance, depending on the parameter type.

Default: Current Time

This command is used if you want to get explicit information on the kinematics. You can get information on the entire set of kinematics, or on each individual kinematic. Use this if your kinematics do not apply directly to object locations and rotations, or if you need this information in your logic. The type parameter specifies the type of information you want, and will be explained shortly. The kinematicindex parameter is optional, and specifies which individual kinematic you want to get information for. For example, if you add a kinematic to travel 5 units in the x direction as the second addkinematic command, you would pass a 2 as the kinematicindex parameter into the getkinematics command. If not passed, or if you pass a 0 value here, the default gets information for all kinematics together. The updatetime/traveldist parameter is optional. The meaning of this parameter depends on the type parameter you specify. Sometimes this parameter will not be used. Some of the time it represents the requested update time that you want to get information for. If not passed, the current time is used. In the case of a KINEMATIC_ARRIVALTIME query, this parameter instead represents travel distance. The use of this parameter will be explained with each query type.

Information	Value of type Parameter	Single Kinematic	All Kinematics
Location	KINEMATIC_X KINEMATIC_Y KINEMATIC_Z	These return x, y, or z component of the current location of the specified kinematic at the given update time. For example, if you added a kinematic to travel 10 units in the x, starting at time 5, and you want to know the x location for this given kinematic at time 7, you can call getkinematics(datanode, KINEMATIC_X, index, 7) to get the x location for time 7.	These return the x, y, or z location of the object at the given update time.
End Distance	KINEMATIC_ENDDIST		This returns the distance from the final destination location of all kinematics from the object's initial location. Here the updatetime parameter is not used.
Total Distance	KINEMATIC_TOTALDIST	This returns the total distance for the kinematic operation.	This returns the sum of the distances of all the added kinematics. This has a subtle difference from **KINEMATIC_ENDDIST.** For example, if your first kinematic travels 10 in

			the x direction, and your second kinematic travels -10 in the x direction, then the enddist value will be 0, whereas the totaldist value will be 20. Here the updatetime parameter is not used.
Total Component Distances	KINEMATIC_TOTALX KINEMATIC_TOTALY KINEMATIC_TOTALZ	These return the x, y, or z component of the total distance for the kinematic operation. These are the same values you passed into the addkinematic command. Here the updatetime parameter is not used.	These return the sum of x, y, or z component of all added kinematics. Here the updatetime parameter is not used.
Cumulative Distance	KINEMATIC_CUMULATIVEDIST		This returns the cumulative travel distance of all added kinematics. Unlike enddist or totaldist, it calculates the distance of the possibly curved path that the object will follow during the entire kinematic operation. Here the updatetime parameter is not used. Note: for cumulative distance, if you are using turn kinematics, the turn kinematics may not overlap other travel or turn kinematics. If they overlap with others, then the cumulative distance calculation will be incorrect.
Velocity	KINEMATIC_VX KINEMATIC_VY KINEMATIC_VZ	These return the x, y, or z component of the current velocity for the specified kinematic at the given update time.	These return the x, y, or z velocity of the object for the given time.
Start Speed	KINEMATIC_STARTSPEED	This returns the start speed for the kinematic.	

		This is the startspeed you specify in the addkinematic command. Here the updatetime parameter is not used.	
Peak Speed	KINEMATIC_PEAKSPEED	This returns the peak speed or "reached speed" for the kinematic. This is usually the same as the target speed specified in the addkinematic command, but may not be if the kinematic cannot get to the target speed given the distance it has to travel. Here the updatetime parameter is not used.	
End Speed	KINEMATIC_ENDSPEED	This returns the end speed for the kinematic. This is usually the endspeed that you specify in the addkinematic command, but may not be if the kinematic cannot decelerate/accelerate to your specified endspeed given the distance it has to travel. Here the updatetime parameter is not used.	
Total Velocity	KINEMATIC_VELOCITY	This returns the total velocity of the kinematic for the given time.	This returns a scalar value of the total velocity for the given time.
Acceleration (Starting)	KINEMATIC_ACC1	This returns the acceleration value used to get from the start speed to the target speed. If the start speed is less than the target speed, then this value will be the acceleration value. Otherwise it will be the negative deceleration value. Here the updatetime	

		parameter is not used.	
Acceleration (Stopping)	KINEMATIC_ACC2	This returns the acceleration value used to get from the target speed to the end speed. If the end speed is less than the target speed, then this will return the negative deceleration value. Otherwise it will return the acceleration value. Here the updatetime parameter is not used.	
Rotation	KINEMATIC_RX KINEMATIC_RY KINEMATIC_RZ	These return the x, y, or z rotation of a rotational kinematic at the given update time.	These return the x, y, or z rotation of the object at the given update time. This will only work if rotations are managed manually.
End Rotational Distance	KINEMATIC_ENDRDIST		This returns the distance from the final destination rotational position of all kinematics from the object's initial rotational position. This will only work if rotations are managed manually. Here the updatetime parameter is not used.
Total Rotational Distance	KINEMATIC_TOTALRDIST	This returns the total rotational distance for the kinematic operation if it is a rotational kinematic.	This returns the sum of the rotational distances of all the added kinematics. This will only work if you are managing rotations yourself. Here the updatetime parameter is not used.
Total Component Rotations	KINEMATIC_TOTALRX KINEMATIC_TOTALRY KINEMATIC_TOTALRZ	These return the x, y, or z component of the total rotational distance for the kinematic operation if it is a rotational kinematic. These are the same values you passed	These return the sum of rx, ry, or rz component of all added kinematics. This will only work if you are managing rotations yourself. Here the updatetime parameter

		into the addkinematic command. Here the updatetime parameter is not used.	is not used.
Cumulatvie Rotational Distance	KINEMATIC_CUMULATIVERDIST		This returns the cumulative rotational travel distance of all added kinematics. This will only work if you are managing rotations yourself. Here the updatetime parameter is not used.
Rotational Velocity	KINEMATIC_VRX KINEMATIC_VRY KINEMATIC_VRZ	These return the x, y, or z component of the current rotational velocity for the specified kinematic at the given time if it is a rotational kinematic.	These return the rotational x, y, or z velocity of the object for the given time.
Total Rotational Velocity	KINEMATIC_RVELOCITY	This returns the total rotational velocity of the kinematic for the given time if it is a rotational kinematic.	This returns a scalar value of the total rotation velocity for the given time. This only works if rotations are managed manually.
Start Times	KINEMATIC_STARTTIME	This returns the time that the specified kinematic will start its operation. This is the same starttime that you specified when you added the kinematic. Here the updatetime parameter is not used.	This returns the lowest start time of all the kinematics. Here the updatetime parameter is not used.
Acceleration Time (Starting)	KINEMATIC_ACC1TIME	This returns the total time the kinematic will spend accelerating/decelerating from the start speed to the target speed. Here the updatetime parameter is not used.	
Arrival Time	KINEMATIC_ARRIVALTIME	In this query, the updatetime/traveldist parameter is used as a	

		requested travel distance for the given kinematic. This returns the time of arrival for a certain sub-distance of a given kinematic. For example, if I've added a kinematic that tells the object to travel 5 units in the x direction, but I want to know how long it will take him to travel just 3 of those 5 units, I can use this query, and pass 3 in as the traveldist parameter.	
Peak Time	KINEMATIC_PEAKTIME	This returns the total time the kinematic will spend traveling at the peak speed. Here the updatetime parameter is not used.	
Acceleration Time (Stopping)	KINEMATIC_ACC2TIME	This returns the total time the kinematic will spend accelerating/decelerating from the target speed to the end speed. Here the updatetime parameter is not used.	
End Times	KINEMATIC_ENDTIME	This returns the time that the specified kinematic will finish its operation. This is the same endtime that is returned from the addkinematic command. Here the updatetime parameter is not used.	This returns the highest end time of all the kinematics. Here the updatetime parameter is not used.
Number of Kinematics	KINEMATIC_NR		This returns the number of kinematics that have been added. Here the updatetime parameter is not used.
Turn Kinematic Data	KINEMATIC_STARTANGLE KINEMATIC_TURNANGLE KINEMATIC_TURNRADIUS	These return the start angle, turn angle, and turn radius of a turn kinematic.	

Kinematic Type	KINEMATIC_TYPE	This returns KINEMATIC_TRAVEL if the specified kinematic is a travel operation, and KINEMATIC_ROTATE if the kinematic is a rotate operation.

void profilekinematics(treenode datanode [, int index])

datanode - This is the blank node for kinematic data.

index - This optional parameter specifies the index of a specific index to print information for.
Default: 0 (all kinematics)

This command prints kinematic information to the output console. The index parameter is optional. If it is not passed, then information will be printed for all kinematics that have been added. If it is passed, then the index variable is an index of the added kinematic you want to print information for.

void deactivatekinematics(treenode datanode)

datanode - This is the blank node for kinematic data.

This command tells the kinematic to not update locations when the updatekinematics command is called. Execute this on reset to free the object to move it around in the ortho view.

Sampler

1. Concepts
2. Example
    - Referencing a Label
    - Referencing a Table
    - Assigning a Transporter
    - Changing Item Routing

Sampler Concepts

Overview

The Sampler is a new feature in Version 7. The Sampler is a referencing tool and is convenient for referencing objects, their variables, labels or other properties. It eliminates some need for writing code and makes model building faster and easier.

The Sampler can be used to sample or reference various things throughout FlexSim, including:

- Nodes/Objects
- Numbers
- Colors
- 3D Shapes
- Images
- Paths
- Strings

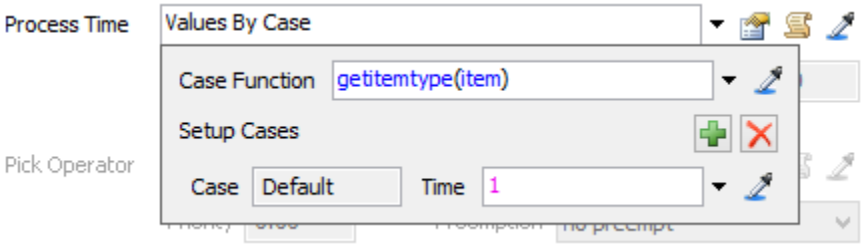
When referencing through the Code Edit window or through picklists, the Sampler will insert correct code for you.

The Sampler is context sensitive, so it will only allow you to sample things in FlexSim that are valid. For instance, if you're trying to specify the Process Time of a Processor, the Sampler will only give you options that return numerical values, like the getlabelnum(), gettablenum, or getvarnum commands. If you are trying to reference an object for the Use Transport option of a FixedResource object, the Sampler will only sample objects and return values like centerobject(current, 1), node("/Operator1", model()).

You can even set values like a label from a Flowitem if it fits the context, simply by sampling the object and choosing the appropriate label from the list.

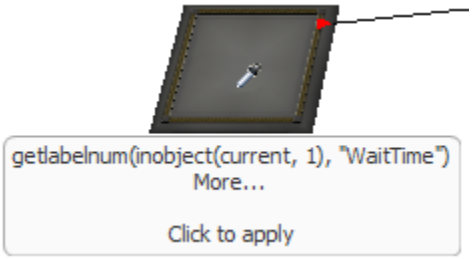
The Sampler is capable of sampling values from the 3D view as well as in Object Properties Windows, Quick Properties, Tree Windows and other windows throughout FlexSim.

You'll find the Sampler in many places throughout the interface. Below is a picklist option for a Processor's Process Time:

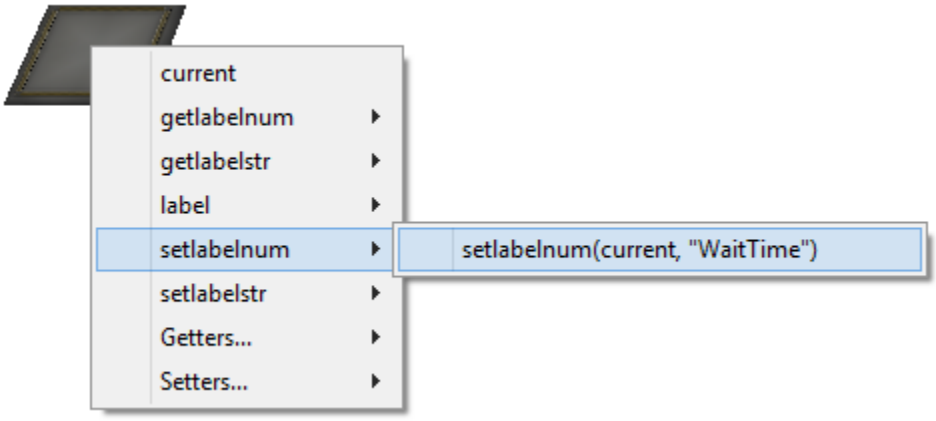


Using the Sampler

To use the Sampler, you simply left-click once on the to enter "Sample Mode". As you move the mouse over objects and fields, the mouse will display available options based on the current context. If an object has no valid data for the current context, no options will be displayed.

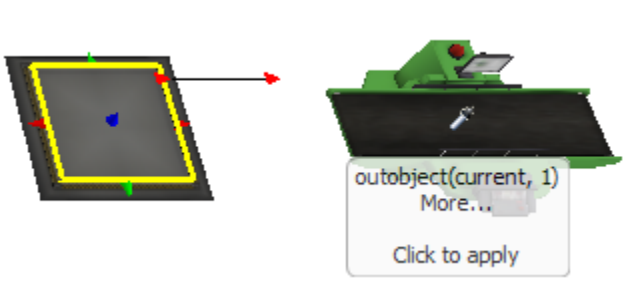


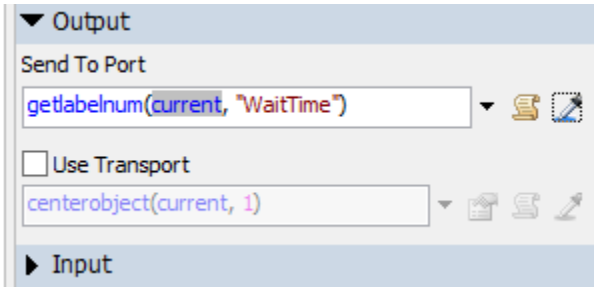
If more than one option is available, a menu will appear allowing you to select the code you wish to use.



Highlighting Code

You can also highlight portions of code you want to change and, instead of figuring out exactly how you're going to reference it, just sample what you need.





Sampling Colors

All of the Sampling functions are restricted to within FlexSim, except for color samplers. If you want to sample a color from another application you can click on a color sampler and click on the desired color outside of FlexSim. Because the mouse is outside of FlexSim, the RGB values will not be displayed under the mouse, however, the Sampler will pull the correct color value under the mouse.

Examples

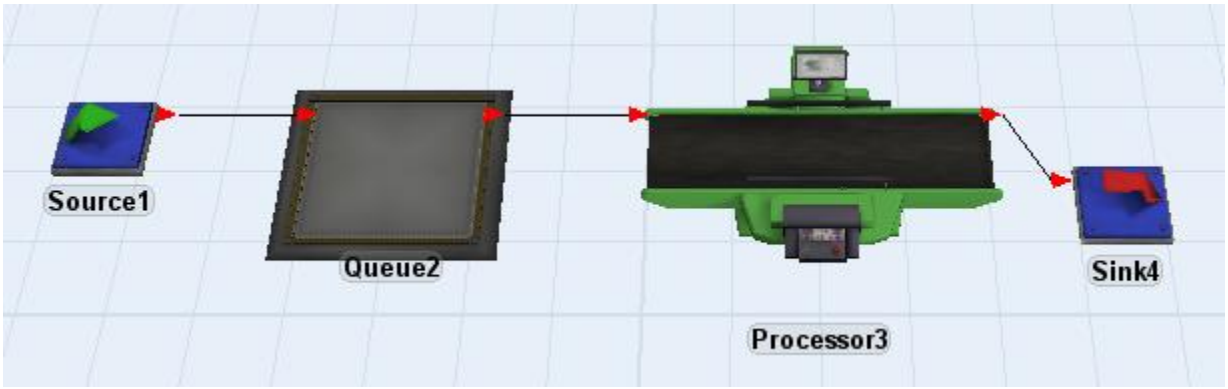
Here are a few examples of where the Sampler is used:

- Referencing a Transporter in an object's flow.
- Defining Setup/Process and Load/Unload Times.
- Choosing object colors.
- Setting object and shape frame 3D shapes.
- Writing code in the Code Editor.
- Referencing objects and numbers in triggers and picklists.
- Referencing Global Table cells.

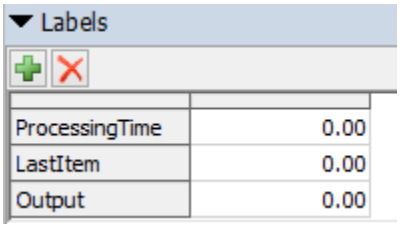
Referencing a Label Example

In this example, we'll show you how to set a processor's **Process Time** dynamically based on a label.

Create a Simple Model



- Create the simple model as shown above.
- Click on the Processor to display it's properties in the Quick Properties window
- Press the + in the **Labels** section of the Quick Properties and add some number labels.

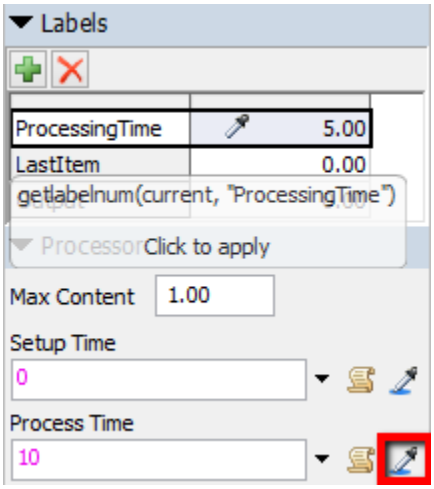


- Set the ProcessingTime label to 5.

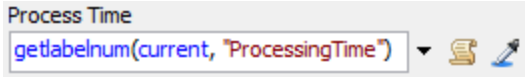
Sample a Label

In the Quick Properties window, below labels, you'll see a **Processor** section.

- Click the next to Process Time to enter "Sample Mode".
- Click on the *ProcessingTime* label you just created to reference that label.



The Sampler will automatically insert the correct line of code for referencing the *ProcessingTime* label.

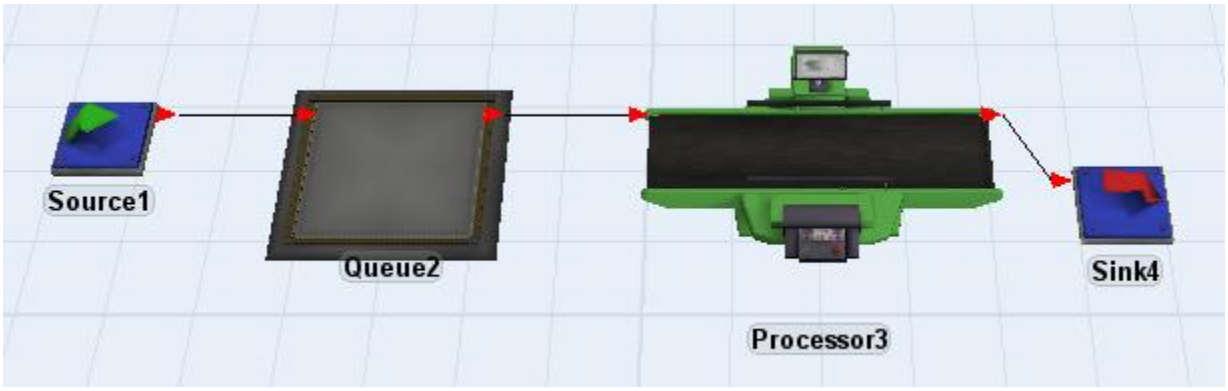


Reset and Run the model to see how it works.

Referencing a Global Table Example

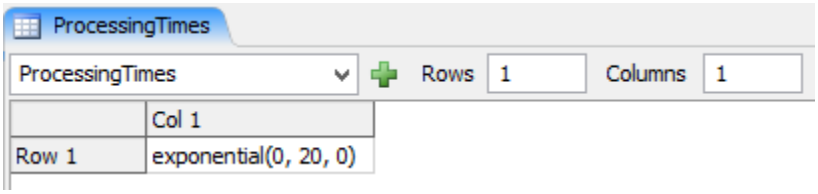
In this example, we'll show you how to set a processor's **Process Time** dynamically based on a cell in a Global Table.

Create a Simple Model



- Create the simple model as shown above.
- Add a Global Table through the Tools menu > Global Tables > Add.
- Change the name of the table to *ProcessingTimes*.
- Right click the first cell and column (0.00) of the Global Table and select **Assign String Data**.
- Enter the following text:

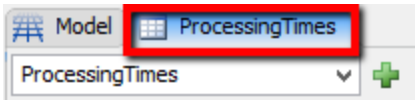
```
exponential(0, 20, 0)
```



Arranging Windows

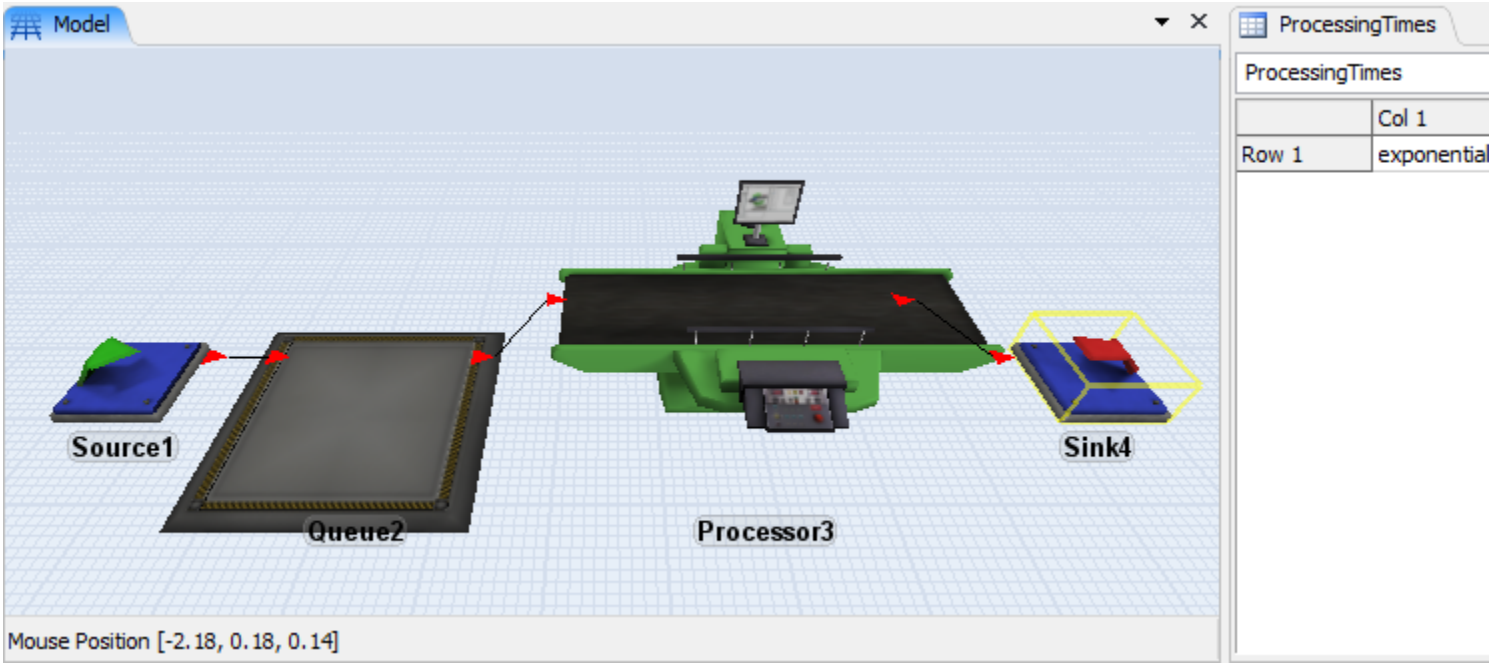
In FlexSim, windows can be arranged in any configuration. Though there are multiple ways to sample a cell in a Global Table, we will look at just one way.

- Click and drag the **ProcessingTimes** table tab to "pull" the Global Table window out of the docked view.



- While still holding the mouse down, drag the mouse over the Dock Windows icon that has appeared in the middle of your main window.

Your main document window will now have your 3D view on the left and your Global Table on the right.

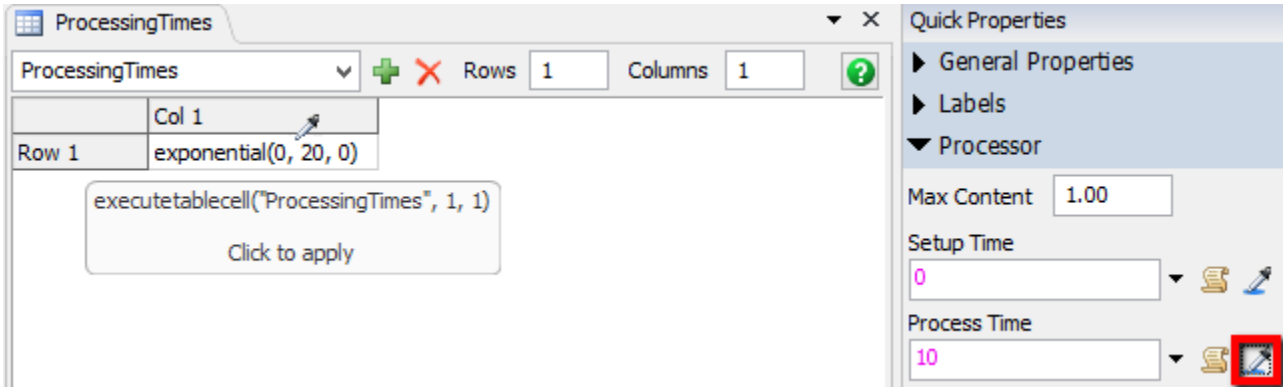


Sample a Global Table Cell

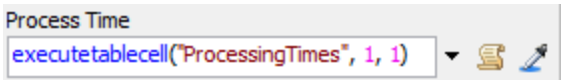
- In the 3D view, click on the Processor object to display its properties in the Quick Properties window.

In the Quick Properties window you'll see a **Processor** section.

- Click the next to Process Time to enter "Sample Mode".
- Click on the first row and column of the *ProcessingTimes* Global Table to reference that cell.



The Sampler will automatically insert the correct line of code for referencing the Global Table cell. Notice that the value the Sampler was looking for was a number (Process Time) and the Global Table cell contained string data. The Sampler automatically added in an executetablecell() command in order to return a numeric value from the cell, in this case exponential(0, 20, 0).

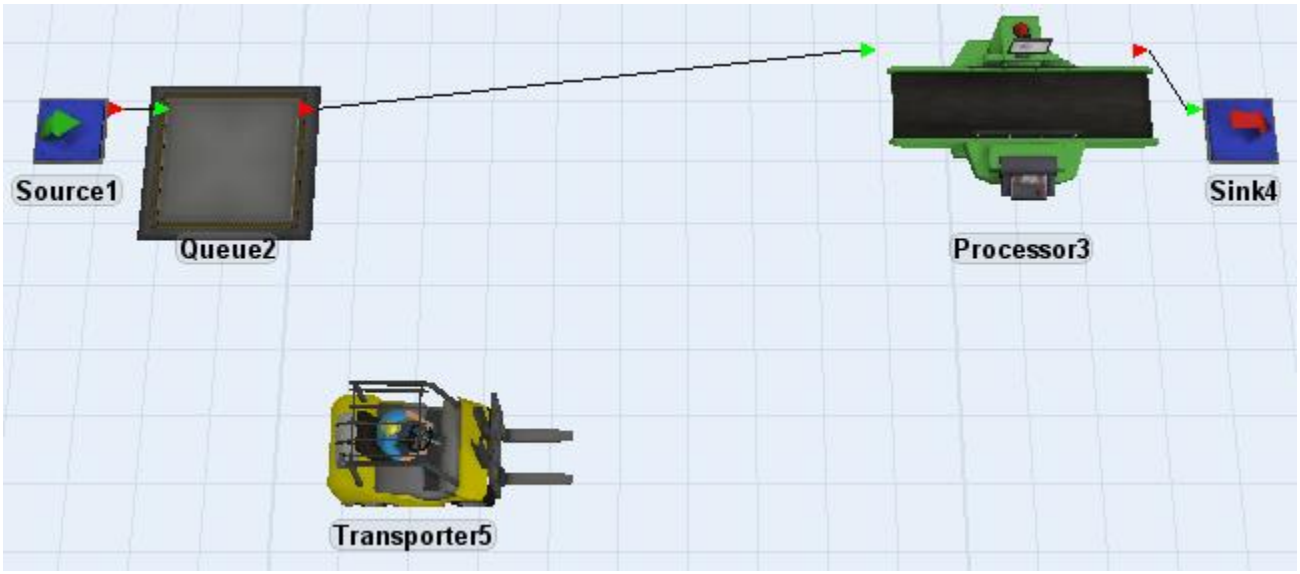


Reset and Run the model to see how it works.

Assigning a Transporter Example

In this example, we'll show you how reference a transporter object to move flowitems from a Queue to a Processor.

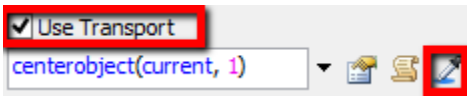
Create a Simple Model



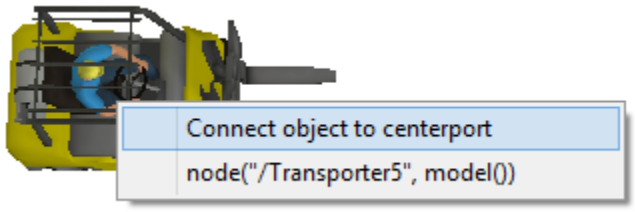
- Create the simple model as shown above.

Assigning a Transporter

- Click on the Queue to display its properties in the Quick Properties window.
- Under the **Output** section of the Quick Properties, check the **Use Transport** button.
- Next to the transport reference picklist, click on the to enter "Sample Mode".



- Click on the Transporter in the 3D view. A menu will appear with the following options:



- Select the **Connect object to centerport** option.

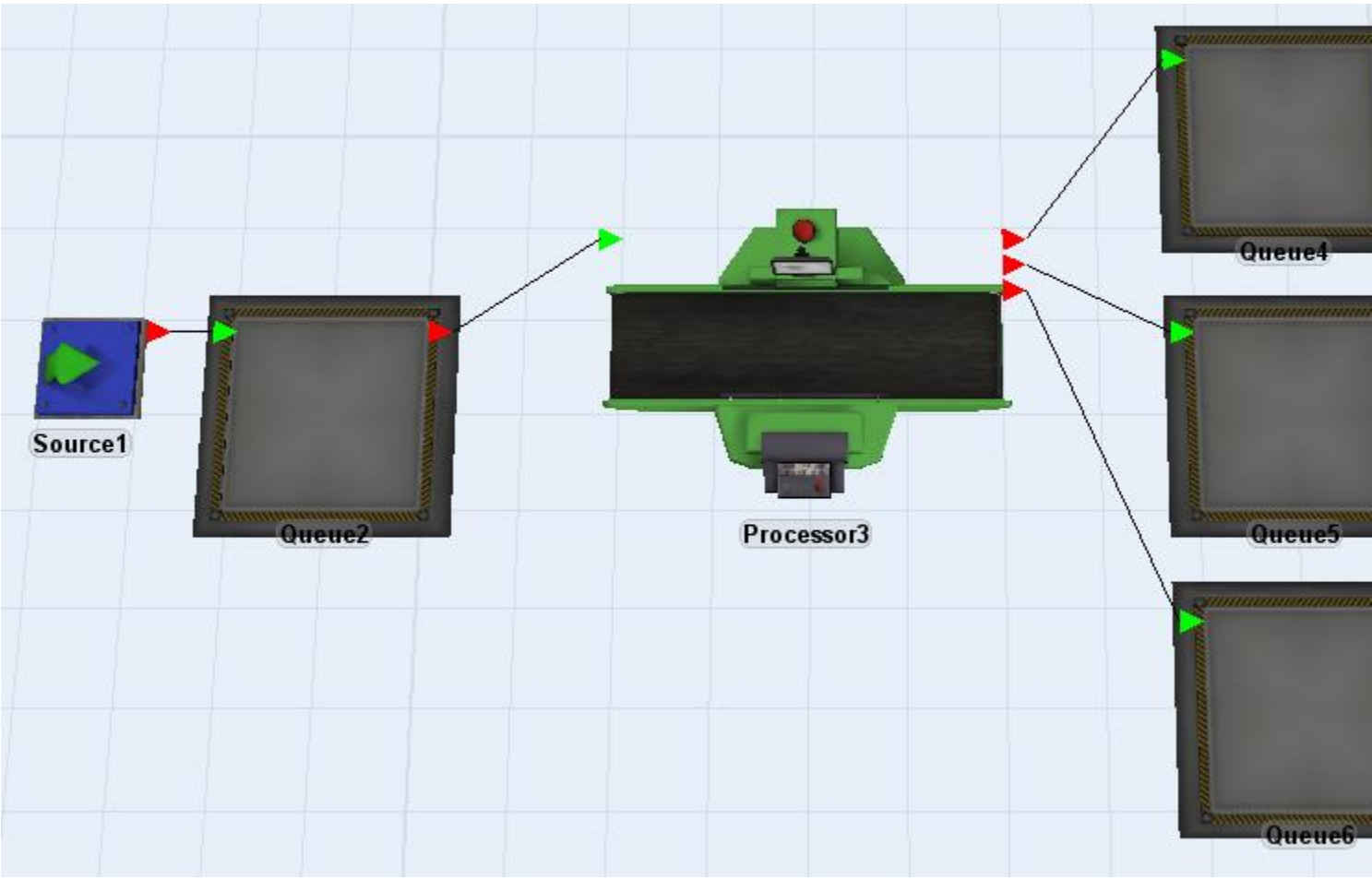
This is will automatically connect the Queue to the Transporter and create the correct reference to the object.

Reset and Run the model to see how it works.

Changing Item Routing Example

In this example, we'll show you how to change a processor's routing based on an item's itemtype.

Create a Simple Model

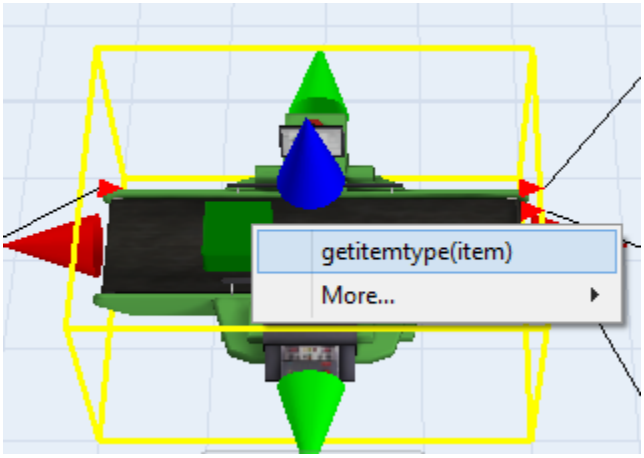


- Create the simple model as shown above.
- Double-click on the Source to open its properties window.
- Go to the Triggers tab and press the ✚ next to the **OnCreation** trigger and select the *Set Itemtype and Color* option.
- Leave the values at their default and press the Ok button on the Source Properties Window to apply the change and close the window.

Sample an Item

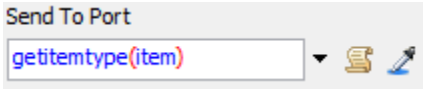
- Reset and run the model until an item appears.
- Stop the model.
- Click on the Processor to display it's properties in the Quick Properties window.
- Under the Output section of the Quick Properties, click the ✎ next to **Send to Port** to enter "Sample Mode".

- Click on the item in your 3D view and the following menu will appear:



- Select the getitemtype(item) option.

The Sampler will automatically insert the correct line of code for referencing the item's *itemtype* into the **Send to Port** picklist.



Reset and Run the model to see how it works.

Note: You may also open the Flowitem Bin and Sample flow items from there. If you are sampling labels on a Flowitem, keep in mind that the options the Sampler will give contain all of the labels on all of your Flowitems. It does not distinguish between flow items.

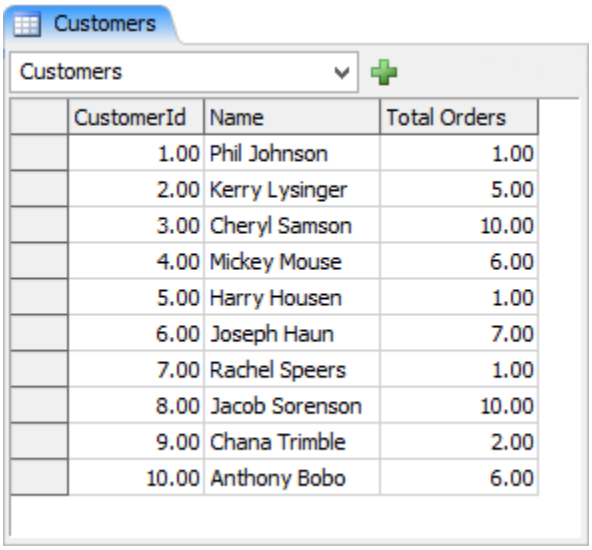
SQL Queries

1. Concepts
2. Example
3. Reference

SQL Queries Concepts

FlexSim provides a flexible method for querying, filtering and prioritizing data in a model using the SQL language. Using SQL queries, users can do standard SQL-like tasks, such as searching one or more tables in FlexSim to find data that matches criteria as well as prioritizing that data. Instead of using FlexScript code to manually loop through and sort table data, you can write a single SQL query using the query() command to do all the work for you.

For example, let's say you have a global table in your model that looks like the following:



	CustomerId	Name	Total Orders
	1.00	Phil Johnson	1.00
	2.00	Kerry Lysinger	5.00
	3.00	Cheryl Samson	10.00
	4.00	Mickey Mouse	6.00
	5.00	Harry Housen	1.00
	6.00	Joseph Haun	7.00
	7.00	Rachel Speers	1.00
	8.00	Jacob Sorenson	10.00
	9.00	Chana Trimble	2.00
	10.00	Anthony Bobo	6.00

And let's say you want to search that table to find the name and id of the customer with the most total orders. The manual method for doing this in FlexScript would look like the following:

```
treenode customers = reftable("Customers");

int bestRow = 0;

int highestTotalOrders = 0;

for (int i = 1; i <= gettablerows(customers); i++) {

    int totalOrders = gettablenum(customers, i, 3);

    if (totalOrders > highestTotalOrders) {

        bestRow = i;

        highestTotalOrders = totalOrders;

    }

}

int bestCustomerId = gettablenum(customers, bestRow, 1);

string bestCustomerName = gettablestr(customers, bestRow, 2);
```

Alternatively, using SQL and FlexSim's query() command, you can do it much more simply, as follows:

```
query("SELECT CustomerId, Name FROM Customers \
```

```
    ORDER BY [Total Orders] DESC LIMIT 1");

int bestCustomerId = getqueryvalue(1, "CustomerId");

string bestCustomerName = getqueryvalue(1, "Name");
```

This is a pretty simple example. More complex tasks such as sorting multiple results or searching multiple tables would be much more complex to perform manually in FlexScript, whereas doing those complex searches using SQL is relatively easy once you understand the rules of SQL and how to write SQL queries.

FlexSim's SQL functionality also includes advanced querying techniques. These allow you to do searches on data in the model that are not structured like standard tables. You can search flow items, task sequences, resources, really any data in a model that can be conceptualized into lists.

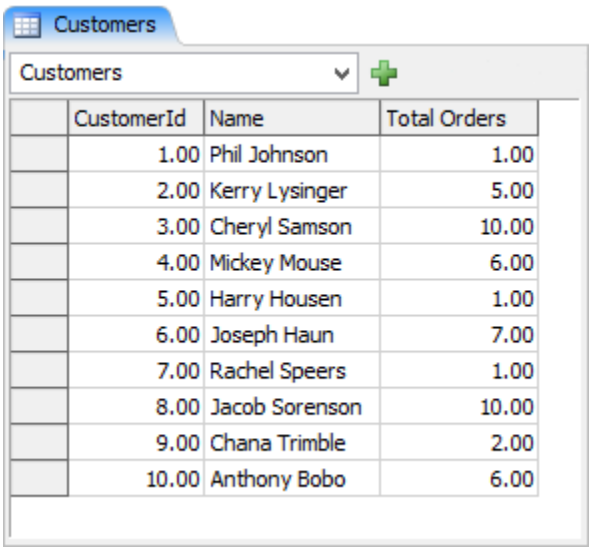
SQL Queries Example

Your First SQL Query

In FlexSim, SQL queries are done using the `query()` command:

```
query(str queryStr[, ...])
```

Often you will use the query() command to search a single table. Take the following example global table.



CustomerId	Name	Total Orders
1.00	Phil Johnson	1.00
2.00	Kerry Lysinger	5.00
3.00	Cheryl Samson	10.00
4.00	Mickey Mouse	6.00
5.00	Harry Housen	1.00
6.00	Joseph Haun	7.00
7.00	Rachel Speers	1.00
8.00	Jacob Sorenson	10.00
9.00	Chana Trimble	2.00
10.00	Anthony Bobo	6.00

Let's say you want to find all customers who have made more than 5 orders, sorted alphabetically by their name. To do that, you'd use the following command:

```
query("SELECT * FROM Customers \
        WHERE [Total Orders] > 5 \
        ORDER BY Name ASC");
```

Some things to note:

- The `SELECT` statement tells the query what columns you want to pull out into your query result. By using `SELECT *` we are telling the query to put all the columns from the source table into the query result. Alternatively, if we were to use `SELECT CustomerId, Name` then that would put just the CustomerId and Name columns into the query result.
- The `FROM` statement defines the table that is to be queried. Often this will be just one table, but it may be multiple comma-separated tables, which effects a join operation, discussed later. Here we define `FROM Customers`. This causes the SQL parser to search the global table in the model named Customers.
- The `WHERE` statement defines a filter by which entries in the table are to be matched. The expression `WHERE [Total Orders] > 5` means that I only want rows whose value in the Total Orders column is greater than 5. In a `WHERE` statement you can use math operators to define expressions, like `+`, `-`, `*`, `/` and logical operators like `=`, `!=` (also `<>`), `<`, `<=`, `>`, `>=`, `AND`, and `OR`.
- The `[ ]` syntax is an SQL delimiter that allows you to define column names with spaces in them. If I had named the column TotalOrders instead of Total Orders, I could have just put `TotalOrders` for the column name in the query instead of `[Total Orders]`.
- The `ORDER BY` statement defines how the result should be sorted. Here we sort by the Name column, which is a text column and thus will be sorted alphabetically. You can optionally put `ASC` or `DESC` to define if you want it sorted in ascending order or descending order. Default is ascending. You can also have multiple comma-separated expressions defining the sort. The additional expressions will define how to order things when there is a tie. For example, the expression `ORDER BY [Total Orders] DESC, Name ASC` would first order descending by the number of total orders, and then for any ties (Cheryl Samson and Jacob Sorenson both have 10 total orders) it would sort alphabetically by name.

- The \ escape character lets you extend a quoted string across multiple lines by using the at the end of a line within a string. Often with SQL queries, the query is too long to reasonably fit within a single-line quoted string. Also, using multiple lines with indentation can make the query more readable.
- A LIMIT statement, although not used in this example, can be added at the end of the query. This will limit the number of matches. If you only want the best 3 matches, add LIMIT 3 to the end of the query.

Getting Data Out of the Query

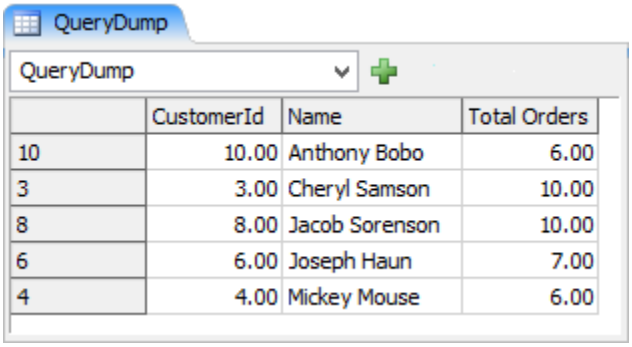
Now that you've done your query, there are several ways to get the results out. The simplest way is to use dumpquery():

```
dumpquery(node toNode, num asTable)
```

dumpquery() will dump the results of the last query() call to a table or bundle node. For this example we can create a global table named QueryDump, and then dump the result to that table. The asTable defines if we want it dumped as table data (1), as opposed to bundle data (0)

```
dumpquery(reftable("QueryDump"), 1);
```

Dumping the above query to the QueryDump table yields the following results:



	CustomerId	Name	Total Orders
10	10.00	Anthony Bobo	6.00
3	3.00	Cheryl Samson	10.00
8	8.00	Jacob Sorenson	10.00
6	6.00	Joseph Haun	7.00
4	4.00	Mickey Mouse	6.00

Table Selection

In the example above, we used "SELECT * FROM Customers". The FROM tells the parser to look in global tables for a table named Customers and search that table. Alternatively, you can explicitly define the tables for search. This is done by using a special '**$**' identifier in the query and by passing additional parameters into the query() command. For example, I could have defined the exact same query as above, but instead defined the customers table explicitly as follows:

```
query("SELECT * FROM $1 \
        WHERE [Total Orders] > 5 \
        ORDER BY Name ASC",
    reftable("Customers"));
```

Notice that instead of using the table "Customers" I now define the table as $1. What this means is that the table I want searched is the table I pass in as the first additional parameter of the query command. $2 would correspond to the table passed into the second additional parameter of the command, and so on. Tables passed as additional parameters can have either regular table data or bundle data; FlexSim will automatically interpret the associated data.

By using this table specification method you are no longer bound to using global tables. For example, if the customers table happened to be on a label instead of a global table, the query is still pretty simple:

```
query("SELECT * FROM $1 \
        WHERE [Total Orders] > 5 \
        ORDER BY Name ASC",
    label(current, "Customers"));
```

FlexSim's SQL parser also allows you to simpify the SQL a bit for this single-table search scenario. The SQL standard requires a SELECT and FROM statement, but FlexSim's SQL parser isn't that picky. If you only pass one parameter as the table, it will automatically assume that you want to search the table you

passed in. Thus you can leave out the SELECT and FROM statements. Leaving them out is essentially the same as using the statement `SELECT * FROM $1`.

```
query("WHERE [Total Orders] > 5 ORDER BY Name ASC", label(current,
"Customers"));
```

Additional Result Analysis Methods

While using `dumpquery()` is often useful when you're setting up a model or when you're testing various queries, filling out an entire table/bundle with data is non-trivial and requires memory space and CPU time for creating the table. Alternatively, if you don't want to use up the resources for creating an entire table, you can directly query the results programmatically using the following commands:

```
getquerymatchcount()
getquerycolcount()
getqueryvalue(num row, num/str col)
getquerymatchtablerow(num/strtable, num nthMatch)
```

As an example, let's say we want to do the query, then manually get the names of each the matching entry.

```
query("SELECT * FROM Customers \
        WHERE [Total Orders] > 5 \
        ORDER BY Name ASC");
for (int i = 1; i <= getquerymatchcount(); i++) {
    string name = getqueryvalue(i, "Name");
    ...
}
```

This performs the query and then goes through each match and gets the name by using `getqueryvalue()` with `"Name"` as the column identifier. You can also access the value by using a number as the column identifier.

```
query("SELECT * FROM Customers \
        WHERE [Total Orders] > 5 \
        ORDER BY Name ASC");
for (int i = 1; i <= getquerymatchcount(); i++) {
    string name = getqueryvalue(i, 2);
    ...
}
```

You can also just access the matching row numbers directly:

```
query("SELECT * FROM Customers \
        WHERE [Total Orders] > 5 \
        ORDER BY Name ASC");
for (int i = 1; i <= getquerymatchcount(); i++) {
    int matchRow = getquerymatchtablerow(1, i);
    string name = gettablestr("Customers", matchRow, 2);
    ...
}
```

In this case we call `getquerymatchtablerow(1, i)`. This returns the row of table 1 (the first and only table defined in the search) associated with the `i`th match of the query.

Joins

You can also use FlexSim's SQL parser to query relationships between multiple tables. To demonstrate this, we'll do another example using the Customers table and an additional Orders Table.

Customers

Customers

	CustomerId	Name	Total Orders
	1.00	Phil Johnson	1.00
	2.00	Kerry Lysinger	5.00
	3.00	Cheryl Samson	10.00
	4.00	Mickey Mouse	6.00
	5.00	Harry Housen	1.00
	6.00	Joseph Haun	7.00
	7.00	Rachel Speers	1.00
	8.00	Jacob Sorenson	10.00
	9.00	Chana Trimble	2.00
	10.00	Anthony Bobo	6.00

Orders

Orders

	OrderId	CustomerId	SKU
	1.00	3.00	78946XU
	2.00	10.00	9302UR
	3.00	2.00	56472VQ
	4.00	5.00	98233EE
	5.00	2.00	22366RG
	6.00	7.00	78946XU
	7.00	1.00	54493BX
	8.00	4.00	61695QH

In this example we want to find information associated with customers who ordered SKU 78946XU. For each order of SKU 78946XU we want to know the customer's name, the CustomerId, and the OrderId. Below is the query:

```
query("SELECT * FROM Customers, Orders \
        WHERE \
            SKU = '78946XU' \
            AND Customers.CustomerId = Orders.CustomerId");
```

Things to note:

- Here we use the statement `FROM Customers, Orders`. In SQL this is called an inner join. The SQL evaluator compares every row in the Customers table with every row in the Orders table to see which row-to-row pairings match the `WHERE` filter.
- The filter we define is `WHERE SKU = '78946XU' AND Customers.CustomerId = Orders.CustomerId`. We only want to match the rows in the Orders table that have the SKU value '78946XU'. Secondly, for those rows that do match the SKU, we only want to match them with rows in the Customers table that correspond with the same CustomerId in the matched row of the Orders table.
- Notice that for the SKU rule we just say `SKU = '78946XU'`. For SKU, since the Orders table is the only table with an SKU column, the SQL evaluator will automatically recognize that the SKU column is associated with the Orders table. We could explicitly define the table and column with `Orders.SKU`, and sometimes that is preferable in order to make the query more readable/comprehensible. However, if you leave it out the evaluator will happily figure out the association on its own.
- The CustomerId rule, on the other hand, uses the `.` (dot) syntax to explicitly define table and column. This is because both the Orders table and the Customers table have columns named CustomerId, and we want to explicitly compare the CustomerId column in Customers with the CustomerId column in Orders. So we use the dot syntax to define table and column.

The result of the dumpquery() for this query:

QueryDump

QueryDump

	CustomerId	Name	Total Orders	OrderId	SKU
3, 1	3.00	Cheryl Samson	10.00	1.00	78946XU
7, 6	7.00	Rachel Speers	1.00	6.00	78946XU

For explicitly defined tables (labels for example) you'd use a query like:

```
query("SELECT * FROM $1 AS Customers, $2 AS Orders \
        WHERE \
            SKU = '78946XU' \
```

```
        AND Customers.CustomerId = Orders.CustomerId",
label(current, "Customers"),
label(current, "Orders"));
```

Aliases

In the previous example we use the `AS` construct to create an alias for our table. You can create aliases for both tables and column references. This can increase readability of the query especially if you are using the `$` syntax. For table aliases you do not technically need the `AS` qualifier. Both of the following are valid:

```
SELECT * FROM $1 AS Customers, $2 AS Orders
SELECT * FROM $1 Customers, $2 Orders
```

Once an alias is defined you should use that alias instead of the table name in other references in the query.

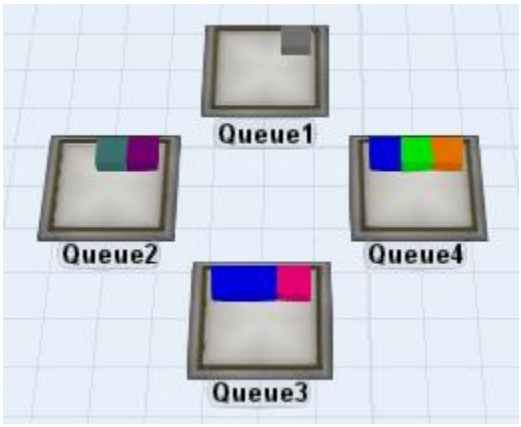
For column references the AS qualifier is required. Below are several examples of defining column aliases.

```
SELECT Customers.CustomerId AS ID, Customers.Name AS Name
WHERE Customers.CustomerId AS ID > 5 AND ID < 10
```

Advanced Query Techniques

Often there are decision-making problems in a simulation that lend themselves well to using SQL-like constructs, such as searching, filtering and prioritizing. However, since a simulation is an inherently dynamic system, data in the simulation is often not structured in the standard way that a database is structured. FlexSim's advanced SQL querying functionality aims to bridge this gap, allowing modelers to use SQL's flexibility and expressiveness in dynamically querying the state of a model.

Let's take an example where you have flow items that queue at various locations in the model. It is quite often the case that modeling logic will try to search for a "best" flow item, and determining which flow item is the best may involve complex rules. It might assign certain eligibility criteria for each flow item. It could also weigh various things like the distance from some source to the location of the flowitem, the time that the flow item has been waiting in queue, an assigned priority for the flowitem, etc. For these types of problems, the SQL language is quite expressive and makes the problem relatively easy. So what if you could represent all of the candidate flowitems as a kind of quasi-database table, and then use SQL to search, filter, and prioritize entries in that table?



Here there are four queues, each with a set of flow items in it. Imagine each of those flow items has various labels defining data on that flowitem. A table of flow items representing all of that data might look something like the following:

Item	Location	DistFromMe	Priority	Step	ItemType	Time Waiting
GrayBox	Queue1	9.85	3	5	6	5.4

PurpleBox	Queue2	8.5	2	2	8	8.1
TealBox	Queue2	8.5	8	12	5	7.2
OrangeBox	Queue4	12.5	4	1	4	1.2
GreenBox	Queue4	12.5	3	5	2	4
BlueBox	Queue4	12.5	6	6	3	22.5
PinkBox	Queue3	7.5	3	9	7	12.8
BlueBox	Queue3	7.5	6	10	3	3.4
BlueBox	Queue3	7.5	4	7	3	7.1

If we could represent the model structure in these quasi-table terms, we could use SQL to do complex filtering and prioritizing based on those tables.

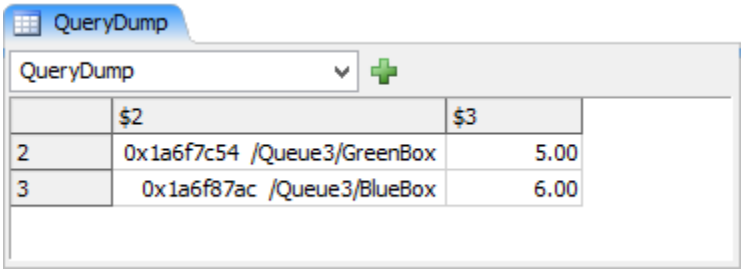
First, let's start simple. We'll just search one queue of flow items, Queue4, and we'll only look at the Step value, which, we'll say is stored on the items' "Step" label. We want to find the flow items with a step value greater than 3. The more simplified table would look like:

Item	Step
OrangeBox	1
GreenBox	5
BlueBox	6

Here's the command.

```
query("SELECT $2 AS Item, $3 AS Step \
            FROM $1 Queue \
       WHERE Step > 3",
     /*$1*/node("Queue4", model()),
     /*$2*/$iter(1),
     /*$3*/getlabelnum($iter(1), "Step"));
```

The results of dumpquery on this would be:



Here we introduce two new concepts: 1. object references as tables, and 2. individual column values being defined by the `$` syntax, instead of just tables.

Object References as Tables

The query table is defined with:

```
FROM $1 Queue
```

And we bind the `$1` reference with:

```
node("Queue4", model())
```

Here instead of referencing an actual table, we reference an object in the model. In doing this, we're associating each row of our "virtual table" with a sub-node of the object we reference, or in other words, a flow item in the queue. So the first row in the table is associated with the first flow item inside the queue, the second row with the second flow item, and so on.

$'s as Table Values and $iter()

The select statement looks like this:

```
SELECT $2 AS Item, $3 AS Step
```

And we bind `$2` with:

```
$iter(1)
```

And `$3` with:

```
getlabelnum($iter(1), "Step")
```

Again here's the "virtual table" we are searching, based on each flow item in Queue4 and its Step label.

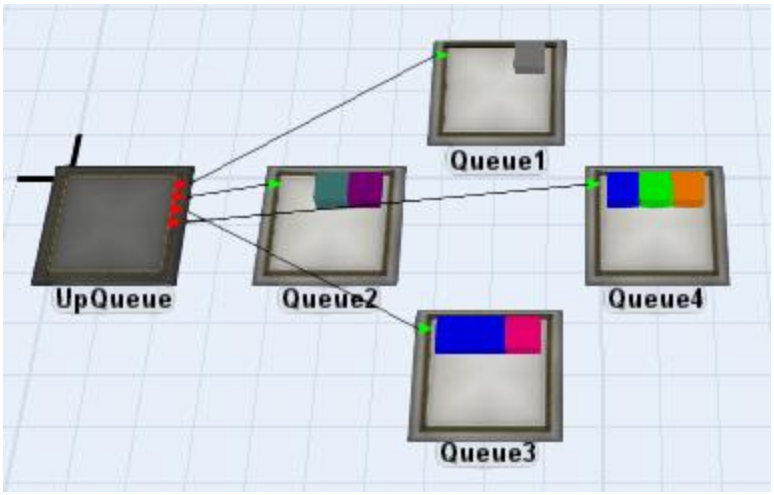
Item	Step
OrangeBox	1
GreenBox	5
BlueBox	6

We use the `$iter()` command to determine the values of the table cells by traversing the content of Queue4. `$iter()` returns the iteration on a given table. `$iter(1)` is the iteration on `$1`. Since `$1` is Queue4, `$iter(1)` is going to be the iteration on Queue4 associated with a given row in the table, or in other words, one of the flow item sub-nodes of Queue4.

When the evaluator needs to get the value of a certain row in the table, it sets `$iter(1)` to the flow item sub-node of Queue4 associated with that table row, and then re-evaluates the expression. So when it's on the GreenBox row in the table (row 2) and needs to get the Step value for that row, it sets `$iter(1)` to `rank(Queue4, 2)`, and evaluates `$3`. `$3` then essentially returns `getlabelnum(rank(Queue4, 2), "Step")`. That value is then used as the value of the table cell.

Creating a Table for All Flow Items

Now that we've done a query on a single queue, let's extend the query to search all queues. Let's assume the queues are all connected to the output ports of an upstream queue.



Now let's write the query.

```
query("SELECT $3 AS Item, $4 AS Step \
                 FROM $1x$2 AS Items \
            WHERE Step > 3",
      /*$1*/nrop(node("UpQueue", model())),
      /*$2*/outobject(node("UpQueue", model()), $iter(1)),
      /*$3*/$iter(2),
      /*$4*/getlabelnum($iter(2), "Step"));
```

The results of dumpquery() on this query are as follows:

QueryDump

QueryDump

	Item	Step
1	0x1a7aeb76 /Queue1/GrayBox	5.00
3	0x1a77b654 /Queue2/TealBox	12.00
4	0x1a6fa27c /Queue3/PinkBox	9.00
5	0x1a7af4be /Queue3/BlueBox	10.00
6	0x1a7b0646 /Queue3/BlueBox	7.00
8	0x1a6f7c54 /Queue4/GreenBox	5.00
9	0x1a6f87ac /Queue4/BlueBox	6.00

Note that we've shifted our column references down. The Item reference is now `$3` and the Step label reference is `$4`. Also notice that we've "flattened" a two-dimensional model structure into a single virtual table by using the special table identifier `$1x$2`. The first dimension is the output ports of UpQueue, and the second dimension is the sub-node tree of each of the Queues connected to the output ports of UpQueue.

Numbers as Tables

For `$1` we return `nrop(node("UpQueue", model()))`. Unlike previously where we assigned `$1` to a Queue object itself, now we define `$1` as a straight number, namely the number of output ports of UpQueue. Subsequently, the iteration on `$1` (`$iter(1)`) will also be a number.

"Flattening" Model Structures

When the SQL parser sees the special `$1x$2` table reference, it will "build" the table by evaluating `$1` and then iterating on `$1`, evaluating `$2` for each iteration. In this example, it evaluates `$1` which returns 3, the number of output ports of UpQueue. Then it iterates from 1 to 3, evaluating `$2` each time. On the first iteration (`$iter(1)` == 1), `$2` returns Queue1: `outobject(UpQueue, 1)`. The evaluator then determines it's an object reference and adds 1 row (the number of flow items in Queue1) to the table. Then it continues to the next iteration on `$1` (`$iter(1)` == 2). `$2` returns Queue2: `outobject(UpQueue, 2)`,

and the evaluator adds 2 rows (the number of flow items in Queue2) to the table, and so on until it's built the entire table. Note that behind the scenes it's not really "building" a table. It's only storing off the total table size and what rows in the table are associated with what `$` iterations. Once it's built the table, the evaluator then goes through the table and evaluates the query for each row in the table, calling `$3` and `$4` to figure out the associated values of each iteration.

In defining a table you can use any number of model structure "dimensions" to define a table, meaning you could have a table that is `$1x$2x$3x$4`, i.e. 4 model-structure dimensions, or more. You can do inner joins on these tables just like you would do with standard tables. Thus, by using these advanced querying techniques you can quickly reduce complex filtering and prioritizing problems into relatively simple SQL queries.

SQL Queries Reference

SQL Language Support

Documenting all of the rules and nuances of SQL is outside the scope of this document. You can get very helpful tutorials on SQL from www.w3schools.com/sql/. FlexSim's SQL parser supports a relatively small subset of SQL language constructs, but hopefully it is enough for most of our users' needs. We list here the FULL set of SQL constructs supported by FlexSim's SQL parser:

- SELECT
  - FROM with inner join using commas and $ syntax to define tables explicitly
- WHERE
  - IN
  - BETWEEN
  - AND/OR
- ORDER BY
  - with multiple comma-delimited criteria
  - ASC, DESC options
- LIMIT
- SQL Aliases using AS
- SQL Functions
  - SUM
  - AVG
  - COUNT
  - MIN
  - MAX
  - STD
  - VAR
  - CNT

SQL Query Result Retrieval Commands

Below are the set of commands you can use in querying and getting query results

`dumpquery(node destTable, num asTable)` - Dumps the full result table for the last call to query() into a node. If asTable is 1, it will dump the result as table data. If asTable is 0, it will dump it as bundle data.

`getquerymatchcount()` - Returns the total number of row in the result table for the last call to query().

`getquerycolcount()` - Returns the number of columns in the result table for the last call to query().

`getqueryvalue(num row, num/str col)` - Returns a value in the result table for the last call to query().
row - The 1-based row number of the result table
col - The 1-based column number of the result table, or the name of the result column. If you have defined aliases in the select statement, the column name should be the alias.

`getquerymatchtablerow(num/str table, num matchRow)` - Returns the source table's row number associated with a matched row in the last query() result.
table - If a string, it is the name of the table as defined in the FROM statement. If you used an alias, it should be the table's alias. If a number, it should be the 1-based index of the table in the FROM statement. If there is only one table in the query, this should be 1. For multiple tables, the first table in the FROM is 1, the second is 2, etc.
matchRow - The 1-based row of the result table of the query.

State List

Below is a list of state numbers and their respective macros. Whenever you write code that has to do with setting the state of objects, like using the stopobject() command or the utilize task, you can substitute these macros in for the number. Refer to the library objects for more information about what each state means to each object.

1 - STATE_IDLE

2 - STATE_PROCESSING

3 - STATE_BUSY

4 - STATE_BLOCKED

5 - STATE_GENERATING

6 - STATE_EMPTY

7 - STATE_COLLECTING

8 - STATE_RELEASING

9 - STATE_WAITING_FOR_OPERATOR

10 - STATE_WAITING_FOR_TRANSPORTER

11 - STATE_BREAKDOWN

12 - STATE_SCHEDULED_DOWN

13 - STATE_CONVEYING

14 - STATE_TRAVEL_EMPTY

15 - STATE_TRAVEL_LOADED

16 - STATE_OFFSET_TRAVEL_EMPTY

17 - STATE_OFFSET_TRAVEL_LOADED

18 - STATE_LOADING

19 - STATE_UNLOADING

20 - STATE_DOWN

21 - STATE_SETUP

22 - STATE_UTILIZE

23 - STATE_FULL

24 - STATE_NOT_EMPTY

25 - STATE_FILLING

26 - STATE_STARVED

27 - STATE_MIXING

28 - STATE_FLOWING

29 - STATE_ALLOCATED_IDLE

30 - STATE_OFF_SHIFT

31 - STATE_CHANGE_OVER

32 - STATE_REPAIR

33 - STATE_MAINTENANCE

34 - STATE_LUNCH

35 - STATE_ON_BREAK

36 - STATE_SUSPEND

37 - STATE_AVAILABLE

38 - STATE_PREPROCESSING

39 - STATE_POSTPROCESSING
40 - STATE_INSPECTING
41 - STATE_OPERATING
42 - STATE_STANDBY
43 - STATE_PURGING
44 - STATE_CLEANING
45 - STATE_ACCELERATING
46 - STATE_MAXSPEED
47 - STATE_DECELERATING
48 - STATE_STOPPED
49 - STATE_WAITING
50 - STATE_ACCUMULATING

Webserver

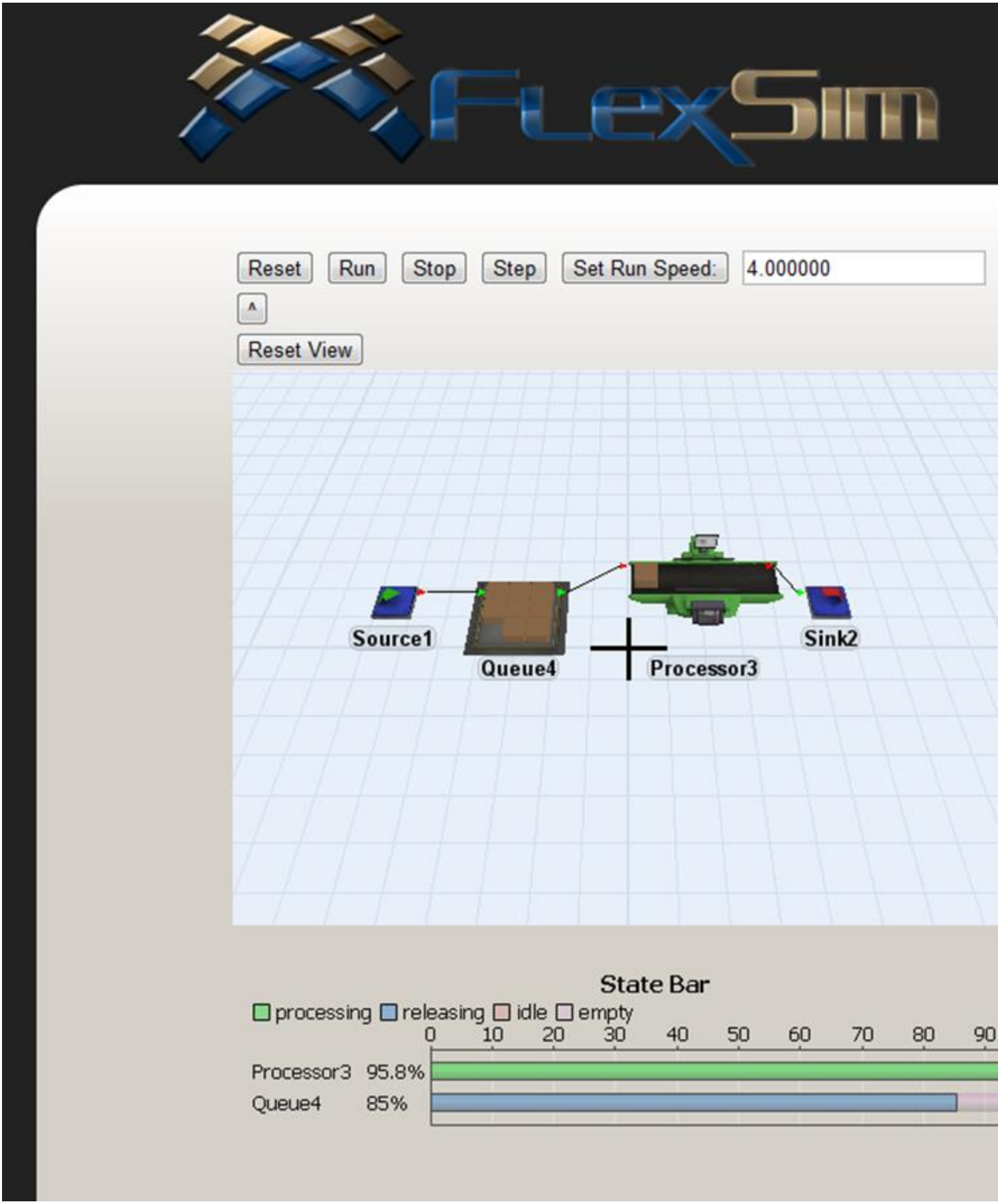
1. Concepts
2. Example

Webserver Concepts

FlexSim''s Webserver is a query-driven manager and communication interface for FlexSim. When you start flexsimserver.exe, your computer will begin to host a website that looks like this:

This website can be accessed by typing the address http://127.0.0.1/ into a browser. You will not have any available models until you put models into your FlexSim Projects directory. The computer in the example has a model called …\Documents\Flexsim 7 Projects\my model.fsm.

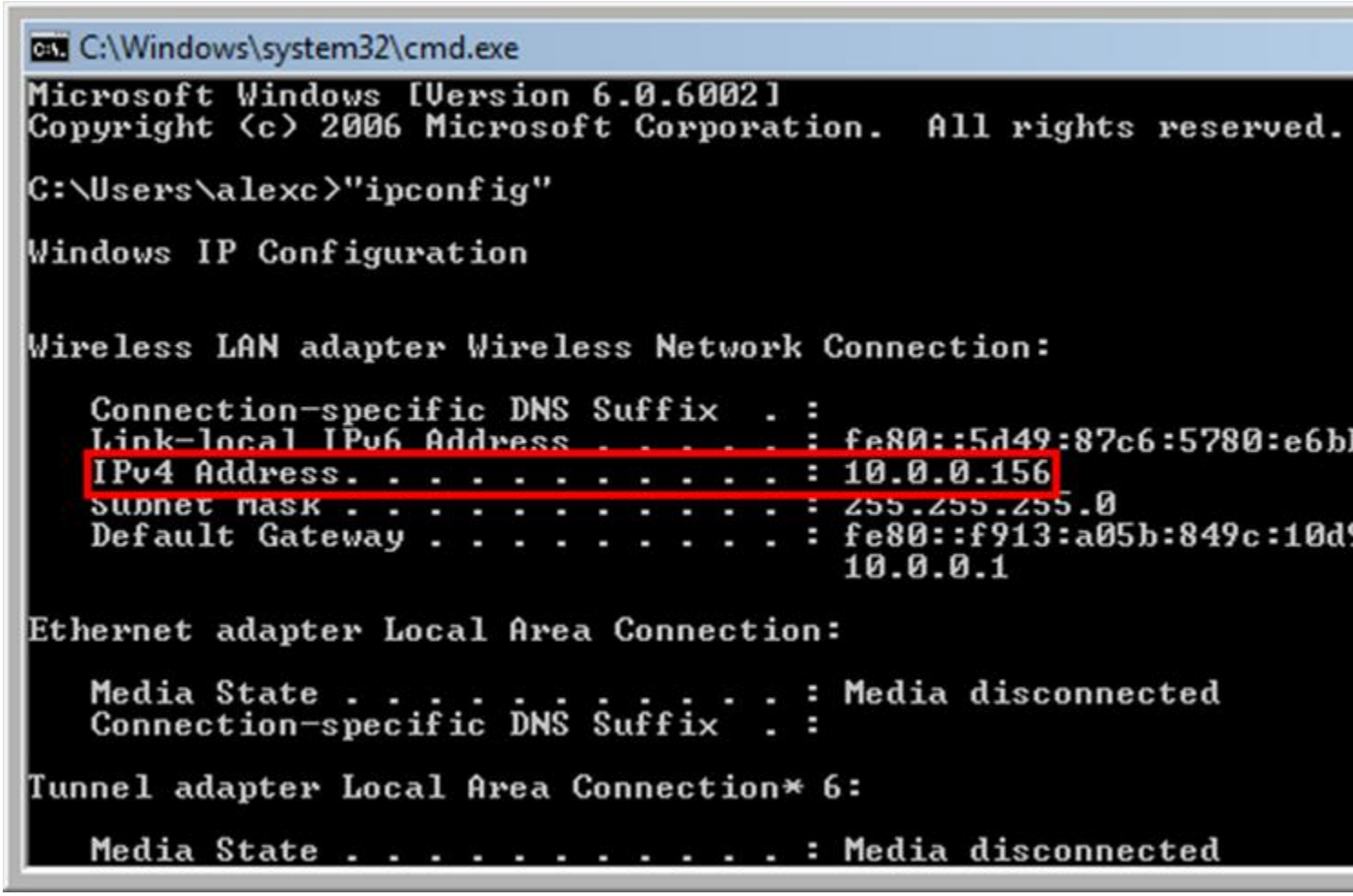
A new instance of FlexSim can be started on the server by pressing the “Start New Instance” button. Once FlexSim starts, click “Connect To Instance” to view the automatically generated web interface for your model. It should look something like this:



You can control your model in the browser using the controls at the top. You can also interact with open views in your model. Left click to pan, right click to rotate, and either right-and-left click or double click to zoom. Unfortunately, browsers do not give Javascript access to the scroll event of an image, so it does not

work like it does in FlexSim. Additional buttons for panning, rotating, and zooming are generated for tablet devices. Click the ^ button to collapse the screenshots and save bandwidth. Graphs from Flexsim's Dashboard are also displayed in real time. If your model has an Experimenter, a standard interface is generated for running experiments.

To connect other devices such as tablets and smartphones to this server, the address 127.0.0.1 will not work. The other device will need to be connected to the same local area network or wireless local area network as your computer. You will need to find the ip address of your computer. To do this, click the start button and type "cmd.exe", which will open a black command prompt. Enter "ipconfig" and press enter. Your IPv4 Address should be entered as the URL in the browser in the device.


C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.
C:\Users\alexc>"ipconfig"
Windows IP Configuration
Wireless LAN adapter Wireless Network Connection:
Connection-specific DNS Suffix . :
IPv4 Address. . . . . : 10.0.0.156
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : fe80::f913:a05b:849c:10d
10.0.0.1
Ethernet adapter Local Area Connection:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . :
Tunnel adapter Local Area Connection* 6:
Media State . . . . . : Media disconnected

To use the server on the internet, though, your computer needs to be assigned a global IP address. Contact your network administrator.

Using Flexsim Webserver with Apache or IIS

flexsimserver.exe has an interface identical to the ISAPI extension webserver.dll, which is in the Flexsim6 program directory. If Apache or IIS are configured to use webserver.dll, they can also manage instances of Flexsim and interact with Flexsim's webserver interface. All interaction should be the same except for two major differences: visibility and threading. Apache and IIS are run in the background, so FlexSim instances will not be visible on the server. This makes it harder to debug any problems. Also, Apache and IIS use multiple threads to respond to requests in parallel, so the order of replying to requests could be different. This makes responding to queries to different instances of FlexSim on the same computer faster with the ISAPI extension than with flexsimserver.exe.

Apache and IIS require some additional configuration to work with FlexSim. The steps are outlined as follows:

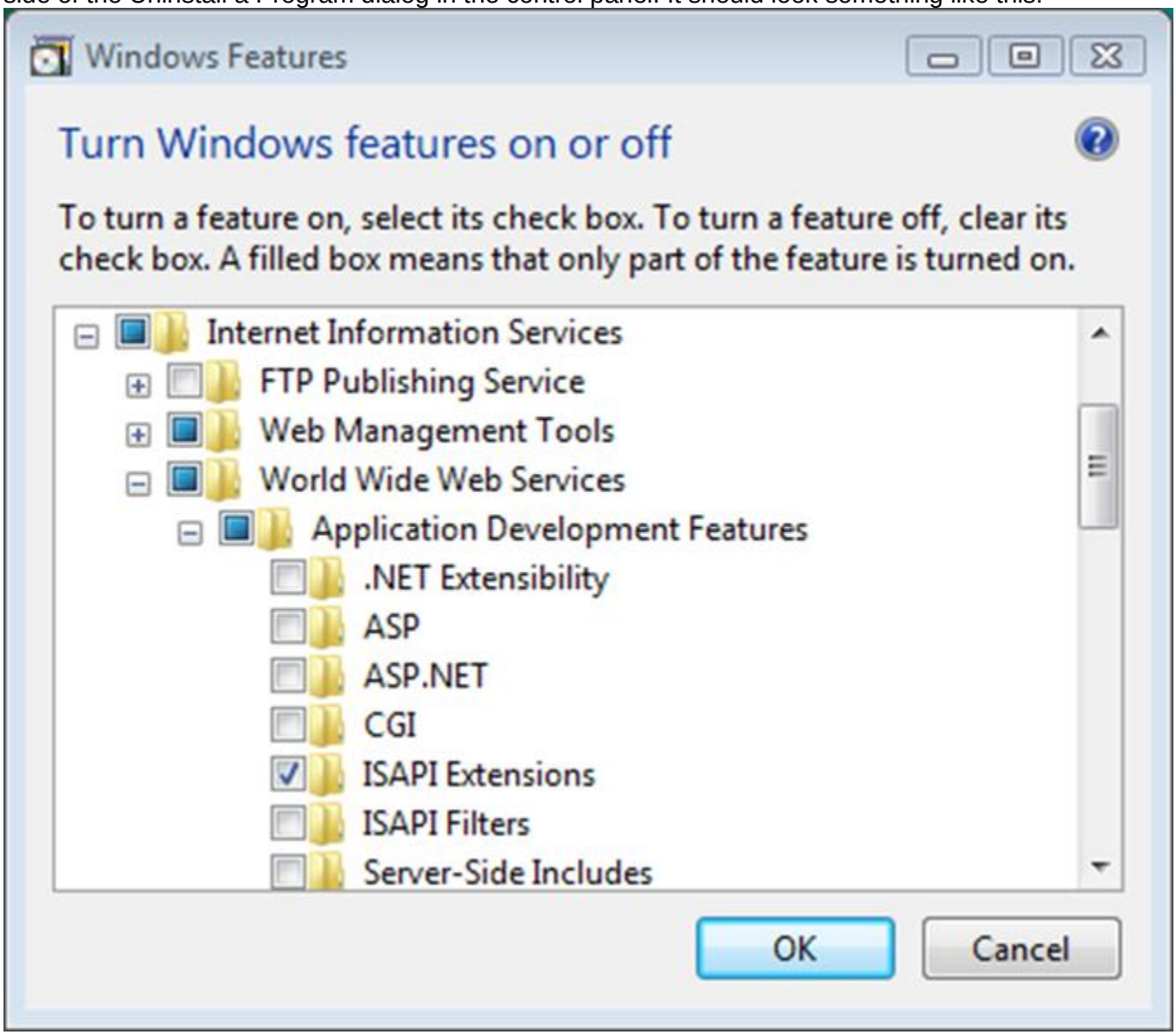
Apache

1. Move flexsim webserver configuration.txt to C:\Program Files\Apache Software Foundation\Apache2.2 or equivalent.
2. Replace any %DOCUMENTS% or %PROGRAMFILESX86% macros in the directories in flexsim webserver configuration.txt with the full path of the directory.
3. Copy webserver.dll from Flexsim6\program, and index.html and the entire flexsimweb folder (not just its contents) from C:\ProgramData\Flexsim\Flexsim6 to C:\Program Files\Apache Software Foundation\Apache2.2\htdocs or to wherever is specified as DocumentRoot in httpd.conf.
4. Add lines similar to these to the end of httpd.conf:
   <IfModule isapi_module>
   AddHandler isapi-handler .dll
   <Directory "C:/Program Files (x86)/Apache Software Foundation/Apache2.2/htdocs">
   Options ExecCGI
   </Directory>
   </IfModule>

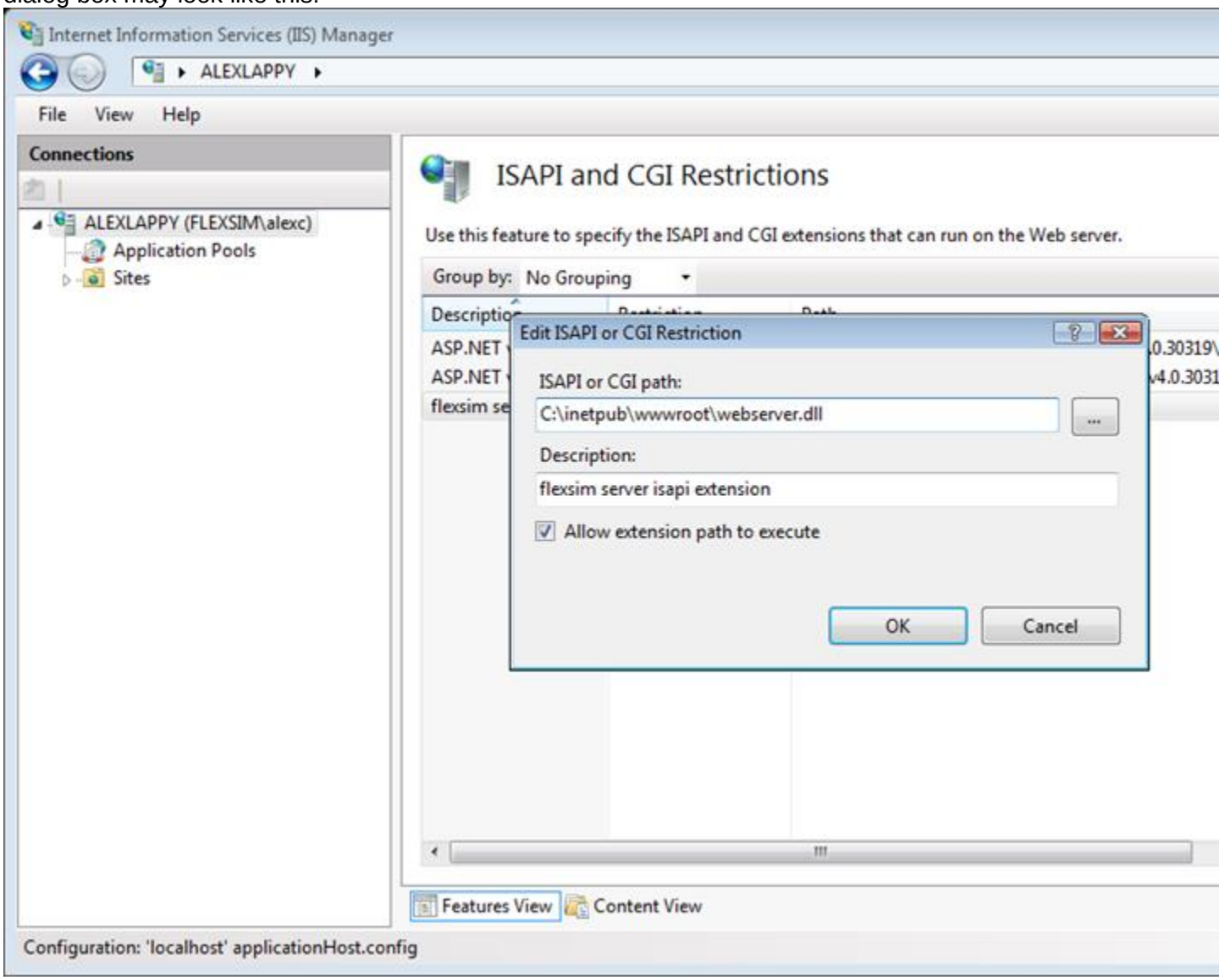
IIS

1. Move flexsim webserver configuration.txt to C:\inetpub.
2. Replace any %DOCUMENTS% or %PROGRAMFILESX86% macros in the directories in flexsim webserver configuration.txt with the full path of the directory. Make sure that Flexsim is installed in a directory that is accessible by IIS and your models are in a directory that is accessible by IIS (for example, not in C:\Users\...).
3. Copy webserver.dll from Flexsim6\program, and index.html and the entire flexsimweb folder (not just its contents) from C:\ProgramData\Flexsim\Flexsim6 to C:\inetpub\wwwroot.

4. Activate ISAPI extensions in the Windows Features dialog. This dialog can be accessed from the left side of the Uninstall a Program dialog in the control panel. It should look something like this:


Windows Features
Turn Windows features on or off
To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.
Internet Information Services
FTP Publishing Service
Web Management Tools
World Wide Web Services
Application Development Features
.NET Extensibility
ASP
ASP.NET
CGI
ISAPI Extensions
ISAPI Filters
Server-Side Includes
OK
Cancel

5. Add an ISAPI extension in the IIS manager. This procedure differs for different version of IIS. The dialog box may look like this:

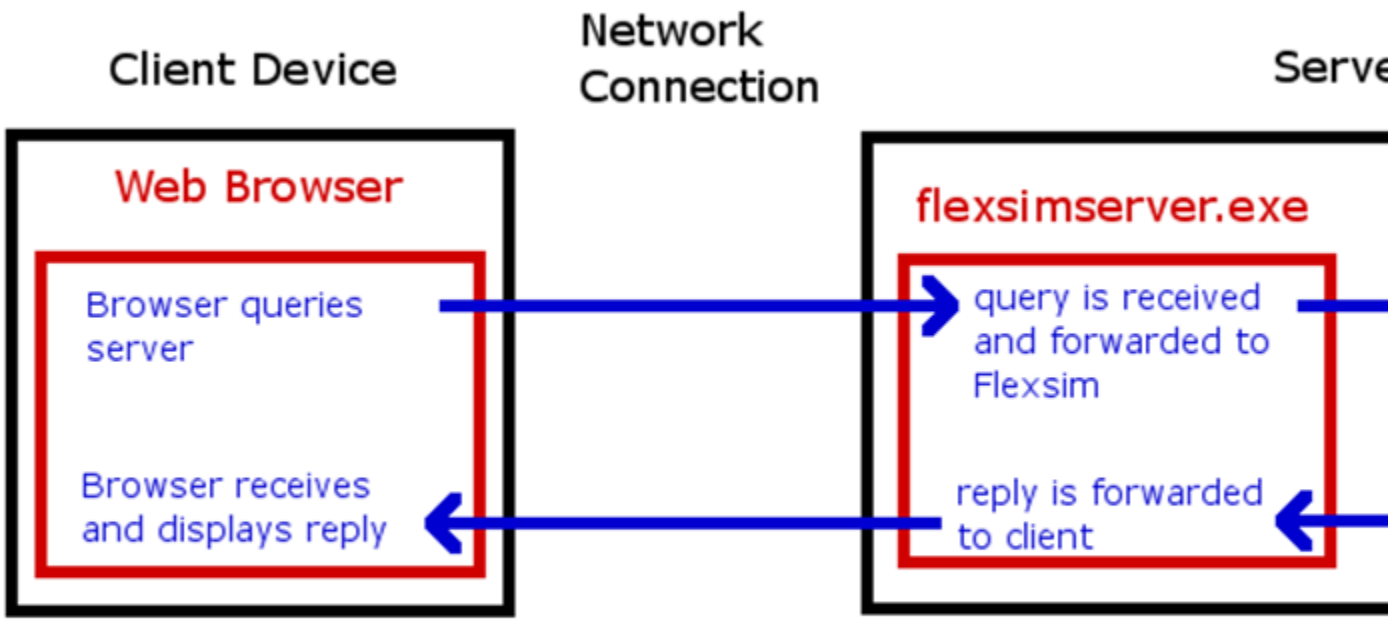


Developing custom web interfaces

flexsimserver.exe and webserver.dll respond to certain types of queries with certain types of actions and responses. For exact details of how they work, try entering the queries into the URL bar of a browser to view the replies. Custom applications can be developed that use these queries:

- webserver.dll?createinstance=my%20model starts an instance of FlexSim running the model specified and returns a short xml reply containing the instance number.
- webserver.dll?terminateinstance=my%model&instancenum=1 terminates the first instance of FlexSim running my model.fsm on the server and returns a short xml reply
- webserver.dll?availablemodels returns an xml list of the models available to run on the server with createinstance queries.
- webserver.dll?instancelist returns an xml list of the models running on the server and their instance numbers. These are the valid possibilities for terminateinstance queries.
- webserver.dll?numinstances returns an xml number representing the number of instances running on the server including from other users. This can be used for load balancing multiple servers.
- webserver.dll?uploadmodel uploads the model to the server. A form such as the form in index.html uploads a model (or another file, such as an excel table) to the model directory. This can be disabled in the flexsim webserver configuration.txt.

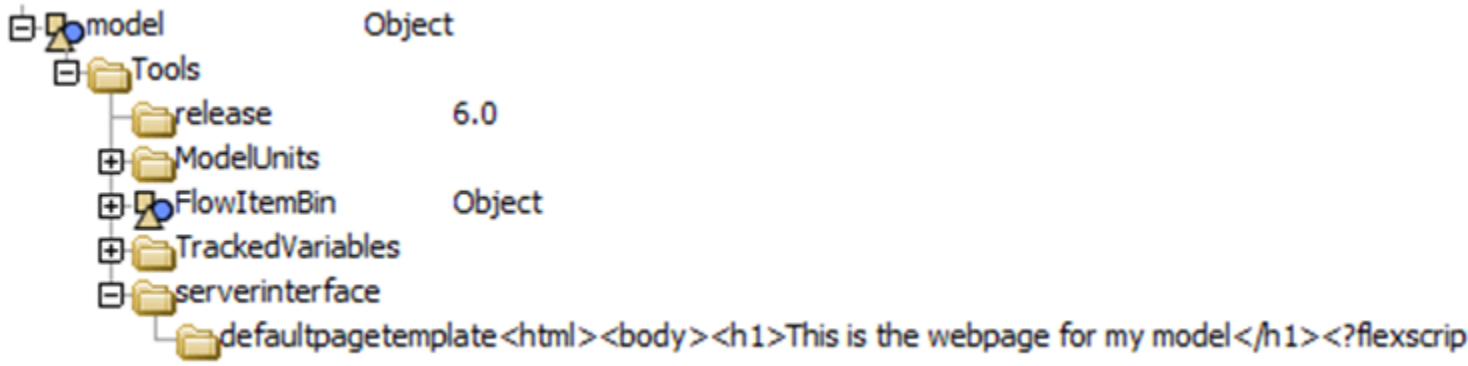
- webserver.dll?deletemodel=my%20model deletes the model (or another file) from the model directory on the server. This can be disabled in the flexsim webserver configuration.txt.
- webserver.dll?allfiles returns an xml list of all files in the model directory on the server.
- webserver.dll?queryinstance=my%model&instancenum=1&... queries an instance of FlexSim. FlexSim decides how to reply to the request. This is the gateway to the main functionality of communicating with a model. The request is handled as seen in this picture:



- webserver.dll?getjobresults=1 returns a JSON summary of a job submitted with a submitjob query.
- webserver.dll?getstatus=1 returns a JSON summary of the job status of a job submitted with a submitjob query. This is a shorter version of a getjobresults query that does not include the full description of the results.
- webserver.dll?getqueuelength returns a JSON number representing the number of jobs in the job queue
- webserver.dll?canceljob=1 cancels a job submitted with a submitjob query if it is incomplete or waiting.
- webserver.dll?getjobquery=1 gets a JSON string such as "queryinstance=my%20model&instancenum=1" that can be used to interact with models run by the job manager. If the job has no running instance (such as if it is waiting or complete), an empty string will be returned because the instance cannot communicate right now. When communicating with an instance of Flexsim started by the job manager, the instance may be shut down at any time. This is why instances started by the job manager are not listed in an instancelist query.
- webserver.dll?submitjob should be the action of an HTML form like this one containing a job description in JSON format. The reply will be an id for use with getjobresults queries.
<form action="http://127.0.0.1/webserver.dll?submitjob" method="POST">
<input type="hidden" name="job"
value='{"modelname":"my%20model","timeout":3600,"priority":5,"setupcommands":[{"command":"settable=modelparameters","data":"values=[[\"firstParam\",0],[\"secondParam\",2],[\"thirdParam\",\"hello\"]]"},{"command":"setrunspeed=100000000"},{"command":"setstoptime=86400"},{"command":"toolsnodefunction=setupscript"},{"command":"run"}],"resultcommands":[{"command":"getnodedata=/Tools/TrackedV

ariables/WorkInProgress"},{"command":"getnodedata=/Tools/modeloutput"}]}'>
    <input type="submit">
</form>
The JSON object must have a "modelname" field, which is the name of the model on the server. It can have a "timeout" field, which is the maximum number of seconds to wait for the model run to complete. It must have a "setupcommands" field, which is an array of command objects. It must have a "resultcommands" field, which is an array of command objects. Command objects must have a "command" field, which is the part of a queryinstance query after webserver.dll?queryinstance=my%20model&instancenum=1&id=1&... Command objects can also have a "data" field, which is the body of the HTTP request, typically containing data from POST requests. Command objects can also have a "verb" field, which is usually "GET" or "POST".
The job will be done as soon as the server is able to make more instances of FlexSim. A createinstance query will be created, followed by a queryinstance=my%20model&instancenum=1&defaultpage query, whose replies are ignored. The replies to each of the setup commands are then stored, followed by polling the model with queries like queryinstance=my%20model&instancenum=1&id=1&getrunstate. When the runstate is 0 (when the model is finished running), the replies to the result commands will be stored. A complete summary of the run in JSON format can be sent as a reply to a getjobresults query.

The default webpage for a model is generated by using a template found in the tree at MAIN:/project/exec/globals/serverinterface/defaultpagetemplate. An html page can be made for a specific model by putting the html into a node created at MAIN:/project/model/Tools/serverinterface/defaultpagetemplate. This page can include the php-like <?flexscript … ?> tags to generate dynamic material. This will override the default page of the model.



Custom query types can be modified or added by adding subnodes to a node at MAIN:/project/model/Tools/serverinterface/queryhandlers. These can be based on the query handlers at MAIN:/project/exec/globals/serverinterface/queryhandlers.

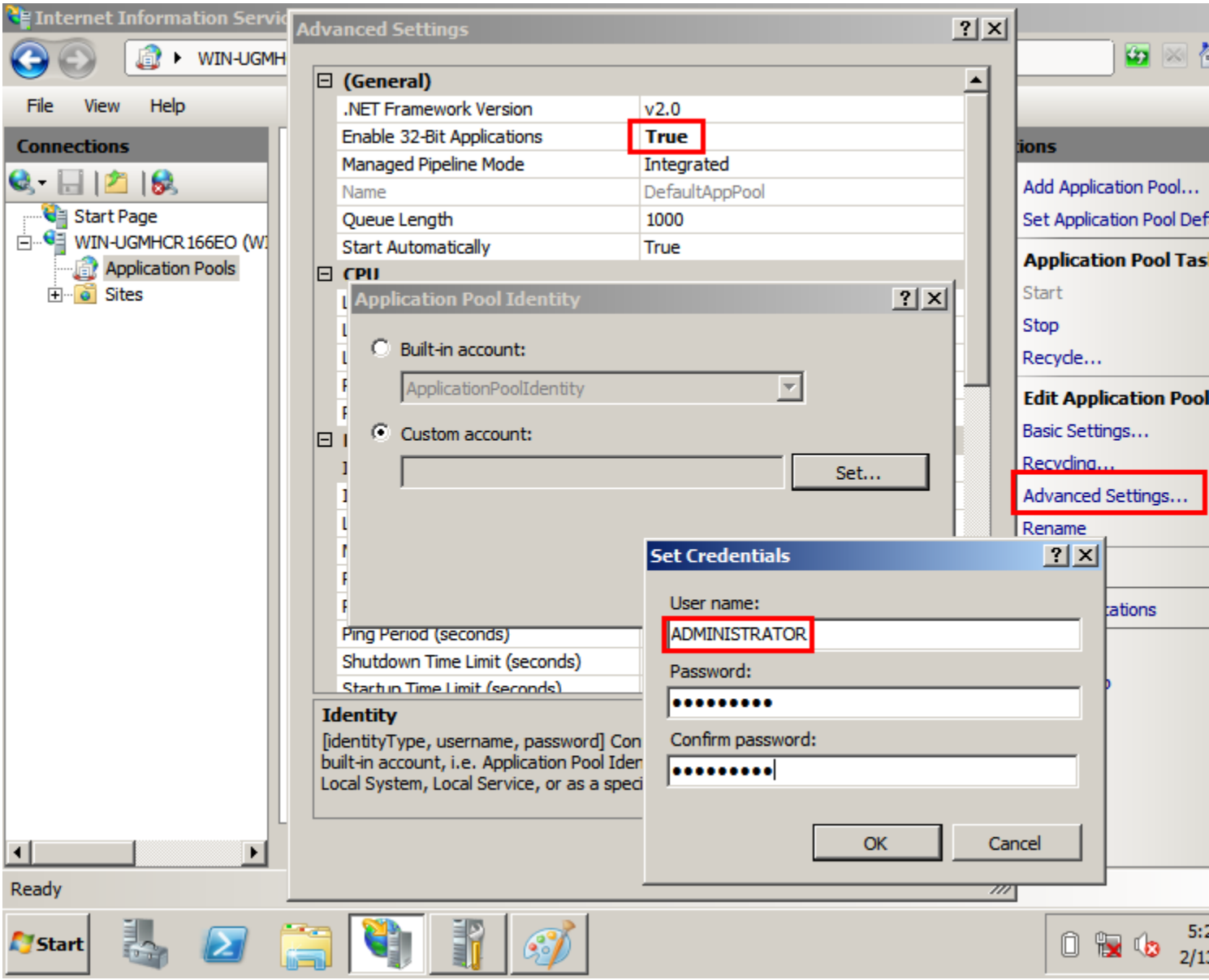
Custom functionality and custom query types for querying the model can also be written for a model by making a flexscript node at MAIN:/project/model/Tools/serverinterface/sendreply. This node must call webcommand("httpsendreply",replynode); once to send a reply to the query. This will send a reply in different forms, depending on the type of data in the node given as a parameter. If the node has string data containing a syntactically correct HTTP reply with HTTP headers as defined by http://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html, then it will be sent as-is, giving developers full access to send any type of HTTP reply desired. Refer to the code in MAIN:/project/exec/globals/serverinterface/sendreply for an example, which can be copied and edited as needed. Remember that the defaultpage and getrunstate queries are used by the job interface.

Webserver Example

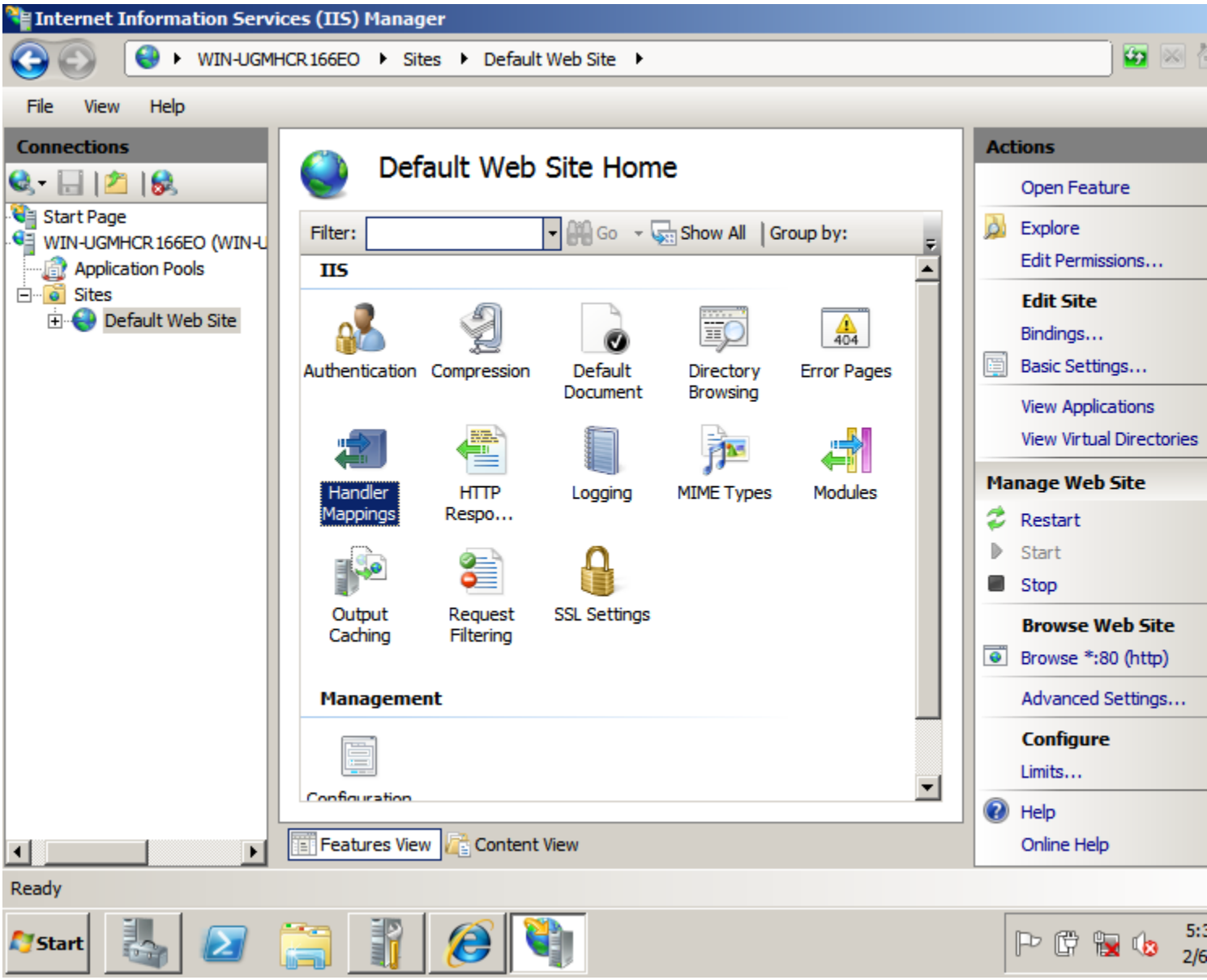
Windows Server 2008 Example

Windows Server operating systems and 64 bit operating systems may have some additional complications. Here is an example of how to get the ISAPI extension working with IIS from a fresh install of Windows Server 2008.

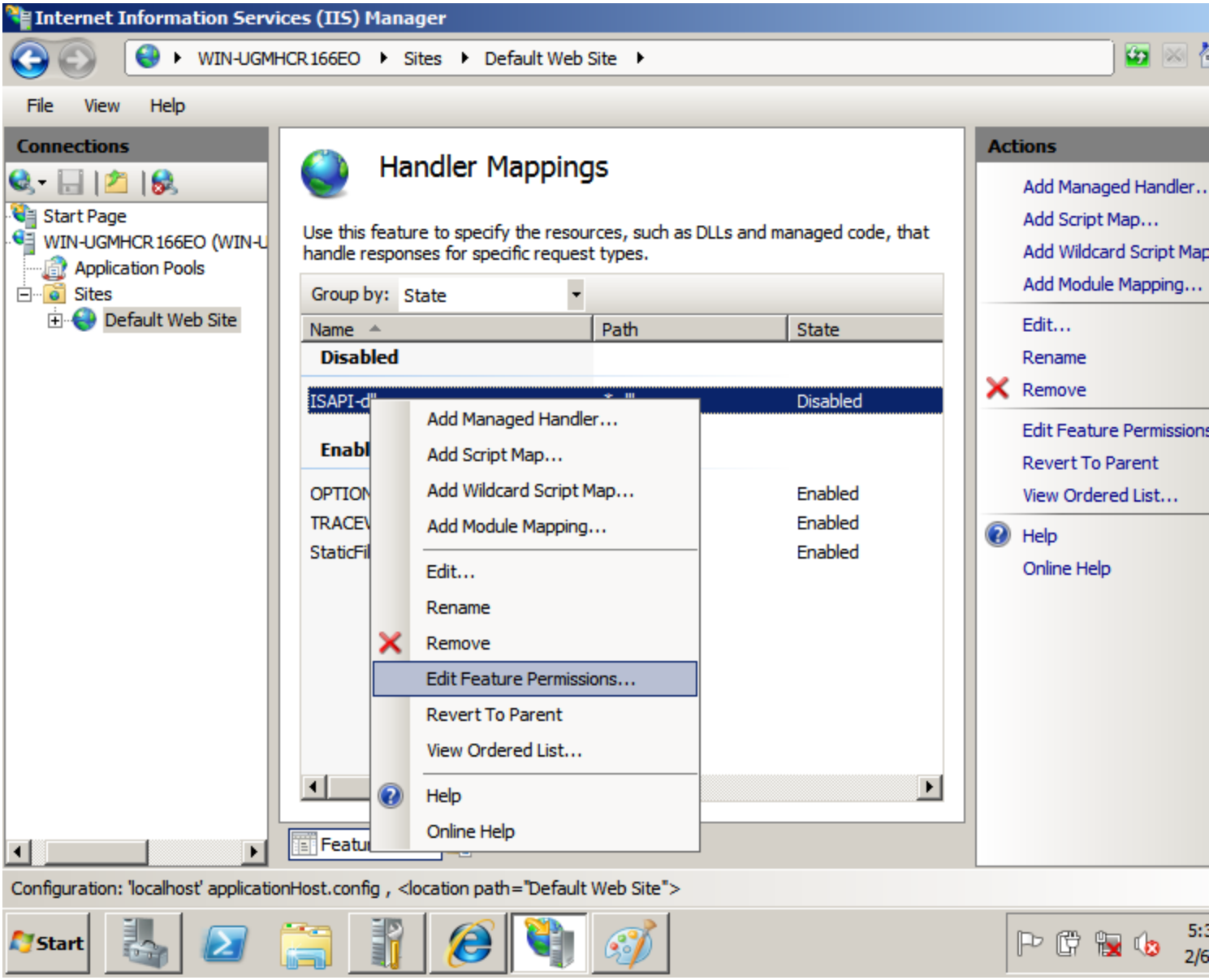
- Open the Server Manager.
- Enable Web Server (IIS) as a new server role.
- Enable ISAPI extensions.
- Install FlexSim. Put flexsim webserver configuration.txt from the FlexSim program folder into c:/inetpub and remove any %...% macros. Put webserver.dll from the Flexsim program folder into c:/inetpub/wwwroot.
- Open the IIS manager and open the ISAPI and CGI Restrictions window. Add a new restriction for C:\inetpub\wwwroot\webserver.dll and allow it to execute.
- Go to the Advanced Settings of the DefaultAppPool, still in the IIS Manager, and click on Advanced Settings. Enable 32-bit Applications and change the Application Pool Identity to ADMINISTRATOR. There will be many permissions issues avoided by doing this.



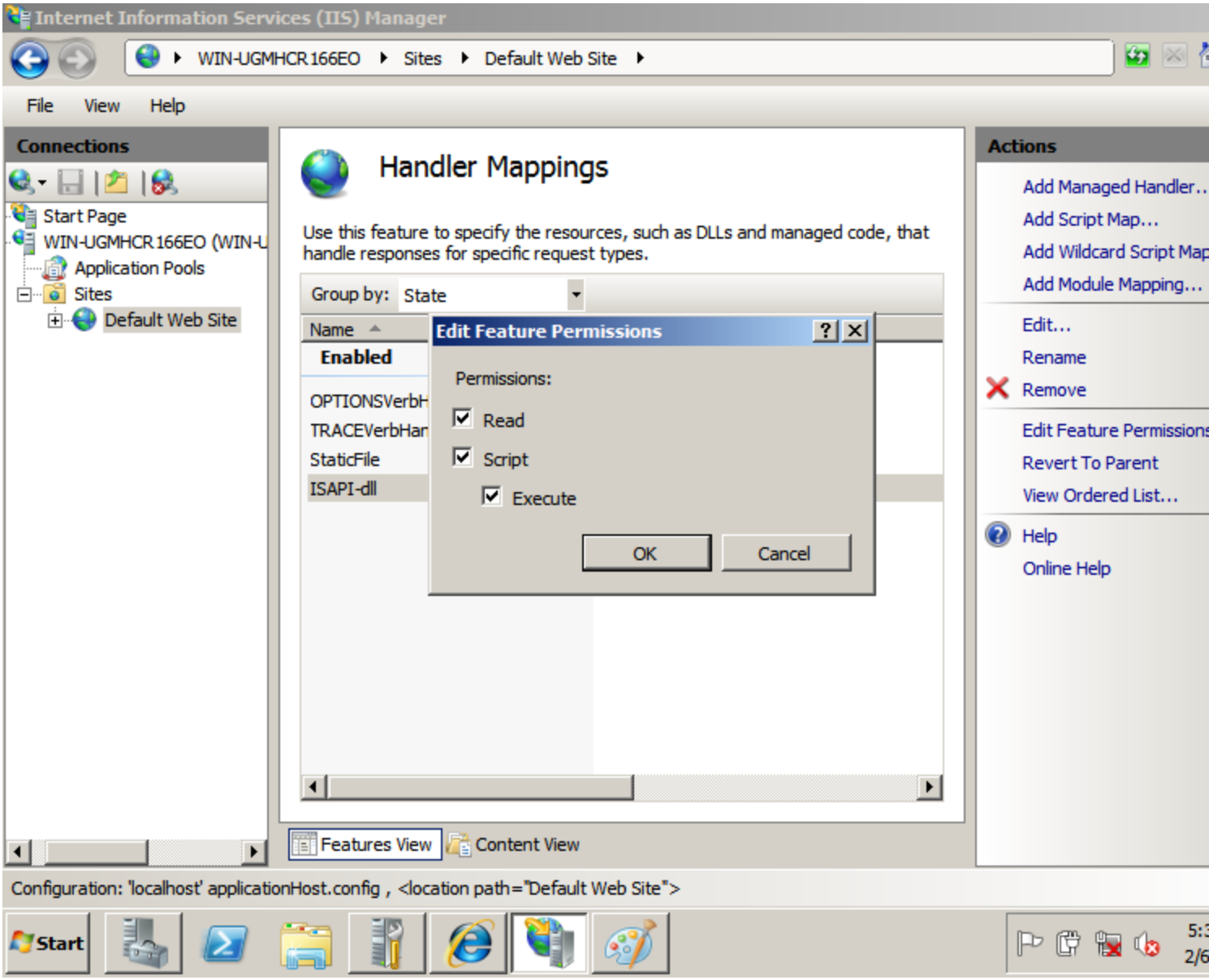
- Under Sites/Default Web Site, double click on Handler Mappings.



- Right click on ISAPI-dll, which is disabled by default, and click Edit Feature Permissions...



- Allow the ISAPI-dll handler mapping to read, script, and execute. Click OK.


Internet Information Services (IIS) Manager
WIN-UGMHCR166EO ▸ Sites ▸ Default Web Site ▸
File View Help
Connections
Start Page
WIN-UGMHCR166EO (WIN-U
Application Pools
Sites
Default Web Site
Handler Mappings
Use this feature to specify the resources, such as DLLs and managed code, that handle responses for specific request types.
Group by: State
Name
Enabled
OPTIONSVerbH
TRACEVerbHar
StaticFile
ISAPI-dll
Edit Feature Permissions
Permissions:
Read
Script
Execute
OK
Cancel
Features View
Content View
Actions
Add Managed Handler...
Add Script Map...
Add Wildcard Script Map
Add Module Mapping...
Edit...
Rename
Remove
Edit Feature Permissions
Revert To Parent
View Ordered List...
Help
Online Help
Configuration: 'localhost' applicationHost.config , <location path="Default Web Site">
Start

When To Compile FlexSim

In general, FlexSim will notify you when you need to compile. If you have set your global preferences to use flexscript code by default, then you should never have to compile unless you open a model that was built using C++.

If you set your preferences to use C++ by default, the following actions will require a compile before running:

- Opening a project or session from the File menu.
- Opening a model that contains any C++ code.
- Creating a new object in the model. This can be done by dragging a new object from the library icon grid into an 3D view. This also can be done by duplicating the selected objects in the model, or creating an object from a user library.
- Adding tools through the Tools menu.
- Editing any pick lists.
- Copying variables from a highlighted object to any set of objects.
- Editing C++ code directly.

Whenever you perform one of the following operations, you may need to reset the model, but you should NOT need to compile the model.

- Editing a regular edit field like the name of an object, or the max content variable of a queue.
- Making connections between objects.
- Editing tables, lists, conveyor sections, etc.

Index

3D Media
Advanced Undo Concepts
Advanced Undo Example
Advanced Undo
Animation Creator Concepts
Animation Creator Example
Animation Creator Reference
Animation Creator
ASRSvehicle Page
Attribute Hints
AVI Maker Concepts
AVI Maker Example
AVI Maker
Basic Modeling Functions
BasicFR Advanced Page
BasicTE Page
Blender Page
Break To
Breakdown Repair Trigger
Breakdowns Page
Breakpoints
Breaks Page
Build Menu
Call Stack
Charting and Reporting
Code Editor
Code Profiler
Collision Page
Collision Trigger
Combiner Page
Command Helper
Container Functionality Page
Container Page
Conveyor Page
Coordinated Task Sequences
Crane Page
Creation Trigger
Custom Built Task Sequences
Custom Chart
Custom Libraries Concepts
Custom Libraries Example
Custom Libraries
Dashboard Associations Page
Dashboard Colors Page
Dashboard Concepts
Dashboard Data Page
Dashboard Date and Time Display
Dashboard Example
Dashboard Financial Objects Page
Dashboard General Pages
Dashboard Graphs
Dashboard HTML Statistic
Dashboard Item Trace Page
Dashboard Objects Page
Dashboard Reference
Dashboard Statistics Page
Dashboard Tracked Variables
Dashboard Utilization Analysis Page
Dashboard
Database Table View
Debug Menu
Debugging Overview
Debugging
Decision Points Page
Dispatcher Page
Display Page
Down Up Trigger
Edit Menu
Edit Selected Objects
End of Experiment
End of Run Replication
End of Scenario
End of Warmup Period
Entry Exit Trigger
Event List
Event Log
Excel Interface
Execute Menu
Experimentation Picklists
Experimenter Optimizer Example
Experimenter Optimizer Reference
Experimenter Optimizer
Experimenter
File Menu
Find and Replace
Find Objects
First Model
FixedResource Pages
FixedResources Concepts
FixedResources Picklists
FixedResources
FlexSim Coding
FlexSim Concepts Overview
FlexSim Concepts
FlexSim Object Library Overview
FlexSim Object Library
FlexSim Terminology
FlexSim Toolbar
FlexSim Tree Structure
FlexSim XML
Flow Page
Flow Rate
Flowitem Bin Concepts
Flowitem Bin Reference
Flowitem Bin
Flowitems

Fluid Library Concepts
Fluid Library
Fluid Objects Step-By-Step Model Construction
Fluid Objects Tutorial Introduction
Fluid Objects Tutorial
Fluid Pages
FluidConveyor Page
FluidLevelDisplay Page
FluidProcessor Page
FluidToItem Page
General Page
General Windows
Generator Page
Geometry Page
Getting Started with FlexSim
Getting Started
Global Modeling Tools Step-By-Step Model Construction
Global Modeling Tools Tutorial Introduction
Global Modeling Tools Tutorial
Global Preferences Window
Global Tables
Global Task Sequences Concepts
Global Task Sequences Example
Global Task Sequences
Global Variables
Graphical User Interfaces Concepts
Graphical User Interfaces Example
Graphical User Interfaces Reference
Graphical User Interfaces
Groups
GUI Events and View Attributes
Help Menu
Importing 3D Media
Importing AutoCAD Drawings
Initial Product
Inputs Outputs Page
Inter Arrivaltime
Interacting With FlexSim
Item and Current
Item Speed
ItemToFluid Page
Itemtype
Keyboard Interaction
Kinematics Commands
Kinematics Concepts
Kinematics Step-By-Step Model Construction
Kinematics Tutorial Introduction
Kinematics Tutorial
Kinematics
Labels Page
Labels Step-By-Step Model Construction
Labels Tutorial Introduction
Labels Tutorial
Labels
Layout Page
Lesson 1 Step-By-Step Model Construction
Lesson 1 Tutorial Introduction
Lesson 1 Tutorial
Lesson 2 Extra Mile Introduction
Lesson 2 Extra Mile Step-By-Step Model Construction
Lesson 2 Extra Mile
Lesson 2 Step-By-Step Model Construction
Lesson 2 Tutorial Introduction
Lesson 2 Tutorial
Lesson 3 Step-By-Step Model Construction
Lesson 3 Tutorial Introduction
Lesson 3 Tutorial
Level Of Detail
Library Icon Grid
License Activation Concepts
License Activation Example
License Activation Reference
License Activation
Light Source Editor
Load Unload Time
Load Unload Time
Load Unload Trigger
Local Variables
Main Menu and Toolbar
Marks Page
Measure Convert
Media Files
MergeSort Flow Page
Message Trigger
Minimum Staytime
Miscellaneous Concepts
Mixer Page
Model Background
Model Input Properties
Model Input
Model Layouts
Model Settings
Model Tree View
Model Triggers
Modeling Tools
Modeling Views
ModelLibraries Node
MTBF MTTR
MultiProcessor Page
NetworkNode Page
NetworkNode
NetworkNodes Page
Node Entry Trigger
Object Properties Windows Overview
Object Windows
On Clear
On Cover
OnChange Trigger
OnCover OnUncover Trigger
OnDraw Trigger

OnEmpty OnFull Trigger
OnEntryRequest Trigger
OnReceiveTaskSequence
OnResourceAvailable Trigger
Optimization in FlexSim
Order of Events
Orthographic Perspective View
Other Picklists
Other Properties Pages
Pass To
Peparing a 3D File
Percents Page
Performance Measure
Photo Eyes Page
Pick Lists
Pick Operator
Picklists
Pipe Layout Page
Pipe Page
Place in Bay
Place in Level
Ports
Presentation Builder
Process Time
Processor Page
ProcessTimes Page
Pull Requirement
Pull Strategy
Querying Information on Task Sequences
Queue Page
Queue Strategy
Quick Properties
Rack Page
Recipe Page
Reports and Statistics
Request Transport From
Reset Trigger
Return Values
Rise Fall Through Mark Triggers
Robot Page
Sample GlobalTable Example
Sample Itemtype Example
Sample Label Example
Sample Object Example
Sampler Concepts
Sampler
Script Console
Send Requirement
Send To Port
Sensors Page
Separator Page
Setup Process Finish Trigger
Setup Time
Shape Factors
Shape Frames
Shared Properties Pages
Simulation Run Panel
Sink Page
SizeTable Page
Source Page
Speeds Page
Split Unpack Quantity
Splitter Page
SQL Queries Concepts
SQL Queries Example
SQL Queries Reference
SQL Queries
SQL Tutorial Introduction
SQL Tutorial Step-By-Step Model Construction
SQL Tutorial
Start of Experiment
Start of Run Replication
Start of Scenario
State List
Statistics Menu
Statistics Window
Steps Page
Table Editor
Tank Page
Task Sequence Preempting
Task Sequence Tutorial 1 Introduction
Task Sequence Tutorial 1 Step-By-Step Model Construction
Task Sequence Tutorial 1
Task Sequence Tutorial 2 Introduction
Task Sequence Tutorial 2 Step-By-Step Model Construction
Task Sequence Tutorial 2
Task Sequence Tutorial 3 Introduction
Task Sequence Tutorial 3 Step-By-Step Model Construction
Task Sequence Tutorial 3
Task Sequence Types
Task Sequences Concepts
Task Sequences Quick Reference
Task Sequences
TaskExecuter Page
TaskExecuter Pages
TaskExecuters Concepts
TaskExecuters Picklists
TaskExecuters
Template Code
Terminator Page
Text Display
Ticker Page
Time Pick Lists
Time Tables Concepts
Time Tables Reference
Time Tables
TimeTables Step-By-Step Model Construction
TimeTables Tutorial Introduction
TimeTables Tutorial

Tools Menu
Tracked Variables
Traffic Control Page
TrafficControl
Transporter Page
Travel Networks Menu
Travel Networks
Tree Browse Dialog
Tree Window
Triggers Page
Triggers
Tutorials Introduction
Tutorials
User Commands Concepts
User Commands Reference
User Commands
User Events Step-By-Step Model Construction
User Events Tutorial Introduction
User Events Tutorial
User Events
View Attributes Reference
View Menu
View Settings
Visio Importer
VisualTool Example
VisualTool Overview
VisualTool
Watch Variables
Webserver Concepts
Webserver Example
Webserver
Welcome To FlexSim
Whats New
When to compile
Writing Logic in FlexSim